

CR-132394

Publicly Released
February 10, 1978

Part I-Final Report, Tasks 1 and 2
**FEASIBILITY STUDY OF AN INTEGRATED
PROGRAM FOR AEROSPACE VEHICLE DESIGN (IPAD)**
Volume IV: IPAD System Design

D6-60181-4
September 21, 1973

(NASA-CR-132394) FEASIBILITY STUDY OF AN
INTEGRATED PROGRAM FOR AEROSPACE VEHICLE
DESIGN (IPAD). VOLUME 4: IPAD SYSTEM
DESIGN Final Report (Boeing Commercial
Airplane Co., Seattle) 367 p HC A16/MF A01 G3/02
N78-16014
Unclas
02568

Prepared under Contract No. NAS1-11441 by
Boeing Commercial Airplane Company
P.O. Box 3707
Seattle, Washington 98124

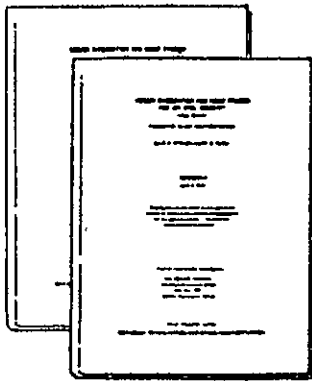
for

Langley Research Center
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION



1. Report No.		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle FEASIBILITY STUDY OF AN INTEGRATED PROGRAM FOR AEROSPACE VEHICLE DESIGN (IPAD) VOLUME IV - IPAD SYSTEM DESIGN				5. Report Date September 21, 1973	
				6. Performing Organization Code	
7. Author(s) W. Goldfarb D. D. Redhed L. O. Anderson L. C. Carpenter S. D. Hansen A. S. Kawaguchi				8. Performing Organization Report No. D6-60181-4	
				10. Work Unit No.	
9. Performing Organization Name and Address Boeing Commercial Airplane Company P. O. Box 3707 Seattle, Washington 98124				11. Contract or Grant No. NAS1-11441	
				13. Type of Report and Period Covered Contractor Report	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546				14. Sponsoring Agency Code	
15. Supplementary Notes Project Manager, Dr. R. E. Fulton, Structures and Dynamics Division, NASA Langley Research Center, Hampton, Virginia 23365					
16. Abstract Volume IV of the Boeing report on Task 2 of the IPAD feasibility study is a description of the computing system design. The requirements which form the basis for the system design are discussed. The system is presented in terms of a functional design description and technical design specifications. The functional design description gives the conceptual organization of the system. The technical design specifications give the detailed description of the system design using top-down structured programming methodology. Human behavioral characteristics, which specify the system design at the user interface; security considerations; and standards for system design, implementation, and maintenance are also part of the technical design specifications. Detailed specifications of the two most common computing system types in use by the major aerospace companies which could support the IPAD system design are presented. The report of a study to investigate migration of IPAD software between the two candidate 3rd generation host computing systems and from these systems to a 4th generation system is included in this volume.					
17. Key Words (Suggested by Author(s)) Integrated Design System System Design Specification Online Computer System Data Base Management Stored Data Definitions				18. Distribution Statement	
19. Security Class. (of this report) Unclassified		20. Security Class. (of this page) Unclassified		21. No. of Pages 367	
				22. Price ██████	

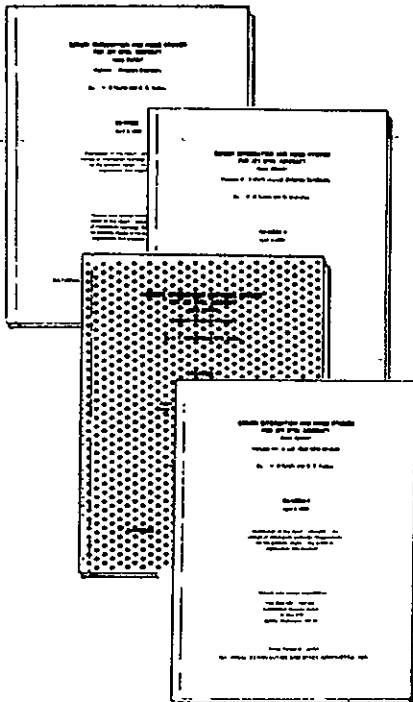
FEASIBILITY STUDY OF AN INTEGRATED PROGRAM FOR AEROSPACE VEHICLE DESIGN (IPAD)



Volume IA
Summary of IPAD Feasibility Study
D6-60181-1A

Volume IB
Concise Review of IPAD Feasibility Study
D6-60181-1B

Part I—Final Report, Tasks 1 and 2



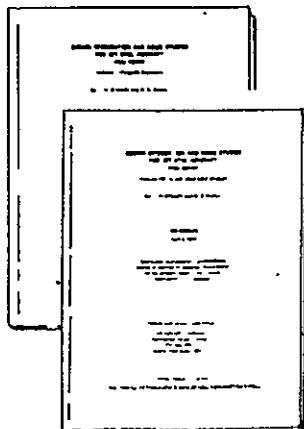
Volume II
The Design Process
D6-60181-2

Volume III
Support of the Design Process
D6-60181-3

Volume IV
IPAD System Design
D6-60181-4

Volume V
Catalog of IPAD Technical Program Elements
D6-60181-5

Part II—Final Report, Tasks 3 through 8



Volume VI
IPAD System Development and Operation
D6-60181-6

Volume VII
IPAD Benefits and Impact
D6-60181-7

SUMMARY

Volume IV describes the IPAD system design. The design is based upon the requirements identified in the aircraft design process and computational requirement studies documented in Volumes II and III respectively. Tables 1 through 4 summarize the relationship of these requirements to the IPAD system design features, the IPAD software requirements and host operating system requirements.

These requirements reflect the user's environment. His tasks are not completed in a day or with a single run on the computer. His interface with the computer should be with language and devices that give him capabilities he needs without loading him with jargon and irrelevancies. He works in large organizations where free communication is essential. But he also works with vast volumes of data that must be controlled and kept in a high state of integrity. The organization he works for has a vested interest in his work and an interest in maintaining some security on the results of his work. At the same time, the user is a creative individual and requires some privacy for thought and invention. The product he is designing is highly complex and he must work under rigid schedules. Reliability of the computing system and the data base is critical. These factors are dealt with in the design of the IPAD system.

The IPAD system is designed to manage data on the project level. Project data and application software are treated as an entry in the data base. The organization of application software into sequences to perform some particular task is supported by executive type routines. The execution of module sequences and the handling of data are supported by the host operating system and the IPAD data manager. Personal terminals are the principal interface and dialogue language is the principal means of communication.

Top-down structured programming is the design method. In this method, the system is systematically refined from the most general statement of requirements to the most specific. The IPAD system design was refined to where host system hardware and operating system software, not yet specified, began to have a major impact.

Human factors, security, and standards were studied in detail and recommendations are given. A survey was made of manufacturers of large scale computing hardware to obtain performance and size characteristics of basic hardware components. The results of this survey were utilized to

formulate a CDC 6600 (CYBER 74) and an IBM 370/168 configuration adequate for a large aircraft design project.

The acceptance of application software already in existence and software that will be developed independent of IPAD system standards was studied by Control Data Corporation. They recommend in their report, included as Appendix C, development of a machine independent FORTRAN language into which the software can be translated.

In this volume, answers to task questions asked in the original RFP from NASA are answered. They are followed by a detailed description of the basic design requirements. The design requirements are then transformed into a functional design that gives a broad diagramatic and conceptual overview of the system. Finally, detailed design specifications of the system are given.

Table 1 IPAD Design Requirement—Continuity Over Task and Time

CONTINUITY OVER TASK AND TIME			
DESIGN REQUIREMENTS	IPAD SYSTEM DESIGN FEATURES	IPAD SOFTWARE REQUIREMENTS	HOST OPERATING SYSTEM REQUIREMENTS
Continuity of day-to-day work	<ul style="list-style-type: none"> • Subtask interruption and restart 	<ul style="list-style-type: none"> • Unique identification of subtasks • Saving/retrieving subtask library • Subtask setup • User log off with job executing 	<ul style="list-style-type: none"> • Time sharing system <ul style="list-style-type: none"> — multi tasking — relationship to IPAD executive — allowable terminal disconnect during execution • Permanent file system
Flow of information throughout the user community	<ul style="list-style-type: none"> • Community library and its associated support routines 	<ul style="list-style-type: none"> • Data display • Information retrieval • Explicit/Implicit I/O • Unique names • Qualifiers • Data management discipline and conventions 	<ul style="list-style-type: none"> • Permanent file system • Data management utilities
Project plans and progress related to the user's day-to-day work	<ul style="list-style-type: none"> • Subtask setup and termination linked to project plans and reports 	<ul style="list-style-type: none"> • Connecting user log-on/off to plans and reports 	
Continuous user capability while migrating across computers	<ul style="list-style-type: none"> • Machine independent high level design 	<ul style="list-style-type: none"> • High level code in the IPAD system written in machine independent source statements 	<ul style="list-style-type: none"> • Compiler for a machine independent language

Table 2 IPAD Design Requirement — User Interface

USER INTERFACE			
DESIGN REQUIREMENTS	IPAD SYSTEM DESIGN FEATURES	IPAD SOFTWARE REQUIREMENTS	HOST OPERATING SYSTEM REQUIREMENTS
Personal Terminal	<ul style="list-style-type: none"> • Unique user ID for each person • Support for typical terminal activities 	<ul style="list-style-type: none"> • User ID tables • Logic to support terminal type activities 	<ul style="list-style-type: none"> • Time sharing system supporting the appropriate type of terminal
Functional Capabilities <ul style="list-style-type: none"> • Define Variables • Enter code and data • Transferring information within IPAD • Sending information outside IPAD • Edit code and data • Purge information • Compare information • Construct jobs for execution • Execute jobs • Display information • Find information • Learning about IPAD 	<ul style="list-style-type: none"> • Defining library entries or variables • Creating library entries • Disposition of library entries • Modifying library entries • Disposition of library entries • Displaying results • Construct an OM sequence as a job • Execute a job • Displaying results • Searching through the libraries • Displaying results • Searching through the libraries • Learning about IPAD 	<ul style="list-style-type: none"> • Executive and data management software • Teaching software 	<ul style="list-style-type: none"> • Executive and data management software support
General Control Commands for <ul style="list-style-type: none"> • Pausing • Continuing • Log-off • Log-on • Assistance 	<ul style="list-style-type: none"> • Interrupted states • Operating system log-off, IPAD log-off • Operating system log-on, IPAD log-on • Learning about IPAD 	<ul style="list-style-type: none"> • Executive logic working with the operating system • Subtask setup and interruption logic • Teaching software 	<ul style="list-style-type: none"> • Roll out, roll in controllable by a user program • Terminal interface for log-on and log-off
Handling of information inside IPAD is invisible to the user	<ul style="list-style-type: none"> • Stored data definition • Coding module, operational module, and job organization 	<ul style="list-style-type: none"> • Support for the stored data definition and automatic library entry handling for constructed jobs 	<ul style="list-style-type: none"> • File assignments changeable by a user program
Human Factors	<ul style="list-style-type: none"> • Interactive dialogue emphasis with helps to aid users at various levels of proficiency 	<ul style="list-style-type: none"> • Interactive logic in the code • Monologue, dialogue and teach modes 	<ul style="list-style-type: none"> • Interactive support to the user • Proper response time characteristics

ORIGINAL PAGE IS
OF POOR QUALITY

Table 3 IPAD Design Requirement — Privacy, Security, Control, and Integrity

PRIVACY, SECURITY, CONTROL, AND INTEGRITY			
DESIGN REQUIREMENTS	IPAD SYSTEM DESIGN FEATURES	IPAD SOFTWARE REQUIREMENTS	HOST OPERATING SYSTEM REQUIREMENTS
Private and public data regions	<ul style="list-style-type: none"> • Subtask library • Community library 	<ul style="list-style-type: none"> • Data management support for this library structure 	<ul style="list-style-type: none"> • Permanent file system • Data management utility routines
Protection against illegal access to information	<ul style="list-style-type: none"> • Data access Permission codes • Command access permission codes • Security control of access codes 	<ul style="list-style-type: none"> • Access code checks • Permission code checks • Specialized procedure for setting codes 	<ul style="list-style-type: none"> • Permanent file system with security lockout • Central memory read/write protection
Assurance of the integrity of the data base	<ul style="list-style-type: none"> • Unique names for all library entries • Mandatory version numbers for all altered library entries • Automatic qualifier generation to record the origin of the data • Trace of information leaving IPAD • Protection against self-inflicted accidents • Setting of permission and access codes • Controlled relationship between the project and its subtasks 	<ul style="list-style-type: none"> • Checks for name uniqueness • Version number generator • Qualifier generator • Keeping records for all information leaving IPAD • Warning about the implications of certain actions • Provision for handling such codes in project plans • Ability to control subtasks on the basis of information in the project plans 	<ul style="list-style-type: none"> • Permanent file names with qualifiers and version numbers

ORIGINAL PAGE IS
OF POOR QUALITY

Table 4. IPAD Design Requirement-Reliability

RELIABILITY			
DESIGN REQUIREMENTS	IPAD SYSTEM DESIGN FEATURES	IPAD SOFTWARE REQUIREMENTS	HOST OPERATING SYSTEM REQUIREMENTS
System unreliability negligible compared to user's unreliability	<ul style="list-style-type: none"> • Recovery of subtask libraries after a system shutdown • Recovery of the entire community library after a system shutdown • Automatic incremental dumps of the system during normal running • Full community library dump capability • Intermittent errors of small effect infrequent • Small error recoverability 	<ul style="list-style-type: none"> • Logic to recover a subtask library that was interrupted out of IPAD's control • Logic to recover a community library directory after a system shutdown • Controls for making incremental dumps at specified intervals • Records of when dumps were taken • Check sums allowing correction 	<ul style="list-style-type: none"> • Recovery of the subtask library file and all files associated with it • Recovery of all permanent files after a system shutdown • Incremental dump feature for the permanent files • Permanent file dump capability • Fault detection hardware • Check sums in all data transfers

CONTENTS

	<u>Page</u>
1.0 INTRODUCTION.....	1
2.0 ANSWERS TO TASK 2 QUESTIONS 1,2,3,12, and 13.....	3
3.0 DESIGN REQUIREMENTS.....	8
3.1 Continuity Over Task and Time.....	9
3.2 User Interface.....	14
3.3 Privacy, Security, Control, Integrity.....	18
3.4 Reliability.....	18
4.0 DEFINITIONS AND ABBREVIATIONS.....	19
5.0 FUNCTIONAL DESIGN DESCRIPTION.....	22
5.1 Primary System Features.....	22
5.2 Work Relationships.....	25
5.3 User Interface.....	27
5.3.1 Personal Terminal.....	27
5.3.2 Command Flow.....	27
5.3.3 Sign On to IPAD.....	28
5.3.4 Communication with IPAD.....	28
5.3.5 Sign Off from IPAD.....	31
5.3.6 Batch Access.....	31
5.4 Libraries.....	31
5.4.1 Library Entries.....	31
5.4.2 Library Variables.....	32
5.4.3 Library Dictionaries.....	32
5.4.4 Library Entry Naming Conventions.....	32
5.5 The Job Concept.....	35
5.5.1 Entering Coding Modules.....	36
5.5.2 Building Operational Modules.....	36
5.5.3 Constructing Jobs.....	37
5.5.4 Execution of Jobs.....	37
5.6 Data Base Management.....	37
5.7 Privacy, Security, Control, Integrity.....	41
6.0 TECHNICAL DESIGN SPECIFICATIONS.....	45
6.1 System Design Methodology.....	45
6.2 System Design Specifications.....	48
6.3 System Member Specifications.....	63
6.3.1 Design Organization.....	63
6.3.2 IPAD Executive.....	64
6.3.3 Data Manager.....	66
6.3.4 Data Structures.....	69
6.3.5 Directory Entry Specifications.....	70
6.3.6 Library Entry Specifications.....	73

6.3.7	Logical Organization of IPAD Libraries in Data Base.....	79
6.4	Human Factors.....	83
6.4.1	User Behavioural Characteristics.....	83
6.4.2	Response Times.....	85
6.4.3	User Classification.....	88
6.4.4	Man-Machine Dialog.....	90
6.4.5	Errors and Failures.....	93
6.4.6	System Balance.....	94
6.5	Security.....	94
6.5.1	Definitions.....	94
6.5.2	IPAD Security Initialization.....	97
6.5.3	Accesses and Requests.....	100
6.5.4	Privacy and Integrity.....	102
6.6	Standards.....	102
6.6.1	IPAD Design Standards.....	103
6.6.2	IPAD Implementation Standards.....	104
6.6.3	IPAD Maintenance Standards.....	106
6.6.4	IPAD Application Standards.....	107
6.7	Language Requirements.....	107
7.0	HOST SYSTEM SPECIFICATIONS.....	109
7.1	Vendor Survey.....	109
7.2	Hardware Characteristics and Capacity Requirements.....	112
7.2.1	Host System.....	112
7.2.2	Terminals.....	118
7.2.3	Networks.....	120
7.3	IPAD Host Computing System Using a CDC 6600 (CYBER 74).....	121
7.4	IPAD Host Computer System Using an IBM 370/168.....	124
	REFERENCES.....	127
	APPENDIX A - DETAILED SYSTEM DESIGN SPECIFICATIONS.....	A1
	LEVEL ONE.....	A7
	LEVEL TWO	
	State E - Subtask Setup.....	A18
	F - Subtask Command Mode.....	A21
	G - Learning About IPAD.....	A26
	H - Searching Through the Libraries.....	A31
	I - Creating Library Entries.....	A35
	K - Modifying Library Entries.....	A38
	M - Constructing a Job.....	A41
	N - Executing a Job.....	A44
	O - Communicating with a Job.....	A47
	P - Displaying Results.....	A50
	Q - Disposition of Library Entries.....	A54

	T - Subtask Step Controlled Abort.....	A58
	U - Subtask Interruption.....	A60
	V - Subtask Termination.....	A63
	W - Defining Library Entries or Variables....	A66
LEVEL THREE		
State	E.A - IPAD Log-On.....	A69
	E.B - Re-Activate Old Subtask.....	A71
	E.C - Create New Subtask.....	A74
	F.A - Request User Input and Interpret Command.....	A77
	F.B - De-Activate Subtask Step.....	A82
	F.C - Re-Activate Subtask Step.....	A84
	H.C - User Controlled Search.....	A87
	H.D - System Controlled Search.....	A90
	I.C - Construct Library Entry.....	A94
	K.A - Connect User with Data to be Modified..	Al00
	K.B - Perform Modifications with Dialog.....	Al04
	M.A - Determine Available Job Components.....	Al10
	M.B - Construct an OM Library Entry.....	Al13
	M.C - Construct a Job Library Entry.....	Al17
	N.A - Establish the Required LEN List.....	Al20
	N.B - Check for LEN in Libraries.....	Al22
	N.C - Prepare Job for Execution.....	Al25
	N.D - Initiate Execution.....	Al28
	N.E - Subtask Step Executing.....	Al30
	Q.C - Purge a CL Entry.....	Al32
	W.C - Construct Dictionary Entry.....	Al36
LEVEL FOUR		
State	I.C.B - Enter Coding Module.....	Al40
	I.C.C - Enter Data Set.....	Al44
	I.C.D - Enter Stored Data Definition.....	Al47
	I.C.E - Enter Dictionary.....	Al51
	I.C.J - Enter Data Control Data.....	Al55
	K.B.A - Modify CM.....	Al58
	K.B.B - Modify OM.....	Al60
	K.B.C - Modify Job.....	Al62
	K.B.D - Modify DS.....	Al64
APPENDIX B - DETAILED PROBLEM SOLVING MODEL.....		
B.1	GENERAL WORK FLOW.....	B1
B.2	"PLAN" NODE DEFINITIONS.....	B5
B.3	"PREPARE" NODE DEFINITIONS.....	B8
B.4	"MODIFY" NODE DEFINITIONS.....	B11
B.5	"WORK" NODE DEFINITIONS.....	B11
B.6	"REPORT" NODE DEFINITIONS.....	B14
APPENDIX C - MIGRATION OF IPAD SOFTWARE.....		
		C1

ILLUSTRATIONS

<u>Figure</u>	<u>Page</u>
1.1	Effect of Information Volume Increase..... 1
3.1	Organizational Hierarachy of Product Design..... 10
3.2	Project Growth..... 11
3.3	Examples of Projects, Tasks, Subtasks and Jobs.... 12
3.4	Phasing of Design Levels..... 13
3.5	General Work Flow..... 15
5.1	Representation of Project Plans..... 23
5.2	Characteristics of IPAD Community and Subtask Libraries..... 25
5.3	Data Base and Work Relationships in IPAD..... 26
5.4	IPAD System Command Flow..... 29
5.5	Sample Job Organization..... 34
5.6	Sequence of Job Definition..... 35
5.7	Steps in Job Construction..... 36
5.8	Data Base Contents..... 38
5.9	Relationships of Implicit and Explicit I/O to The Data Base..... 39
5.10	Attributes of Explicit and Implicit I/O..... 40
5.11	Example of Explicit and Implicit I/O Operations..... 41
5.12	Stored Data Definition Illustrations..... 42
6.1	Structured Programming Diagrams..... 46
6.2	IPAD System Design Level 1 Transition Diagram..... 49
6.3	IPAD System Design Level 1 Transition Diagram (cont.)..... 50

	<u>Page</u>
6.4	IPAD System Member Relationships..... 63
6.5	IPAD Prototype Library Entry..... 69
6.6	IPAD Data Base Library Organization..... 80
6.7	IPAD Library Organization..... 81
6.8	Data Set Library Entry Organization..... 82
6.9	Interactive Response Times..... 87
6.10	Response Time Deviations..... 88
7.1	IPAD Host System - CDC 6600 (CYBER 74)..... 123
7.2	IPAD Host System - IBM 370/168..... 126
APPENDIX A	
A.1	Tree Structure Diagram..... A2
A.2	IPAD System Design Level 1 Transition Diagram..... A5
A.3	IPAD System Design Level 1 Transition Diagram (cont.)..... A6
APPENDIX B	
B.1	General Work Flow..... B2
B.2	Organizational Hierarchy of Product Design..... B4
B.3	An Expansion of PLAN..... B6
B.4	An Expansion of PREPARE..... B9
B.5	An Expansion of MODIFY..... B12
B.6	An Expansion of WORK..... B13
B.7	An Expansion of REPORT..... B15
B.8.1	An Expansion of the Total Work Flow Model..... B17
B.8.2	An Expansion of the Total Work Flow Model. (contd.) B18

TABLES

		<u>Page</u>
2.1	IPAD Development Recommendations.....	6
6.1	IPAD System Design/Volume III Requirements Comparison Summary.....	62
6.2	IPAD System Member Mapping.....	65
6.3	IPAD System Structures.....	74
7.1	Comparison of Vendor Hardware.....	111

1.0 INTRODUCTION

Integrated systems have generally been developed to support a technical analysis requirement without consideration for the information control and communication requirements of the project organization. Within large project organizations, the communication of information between specialized groups is essential. The critical factor in communication is the volume of information being controlled, transmitted or interpreted. As volume increases, response times get longer, reliability deteriorates, control diminishes and information becomes more obscure.

Longer Response Time - In figure 1.1, response time is plotted against volume of information for several transfer rates. There is a band of response times that is effective for a given activity. Response times above this band result in information being transferred too late to be useful to the receiving organization.

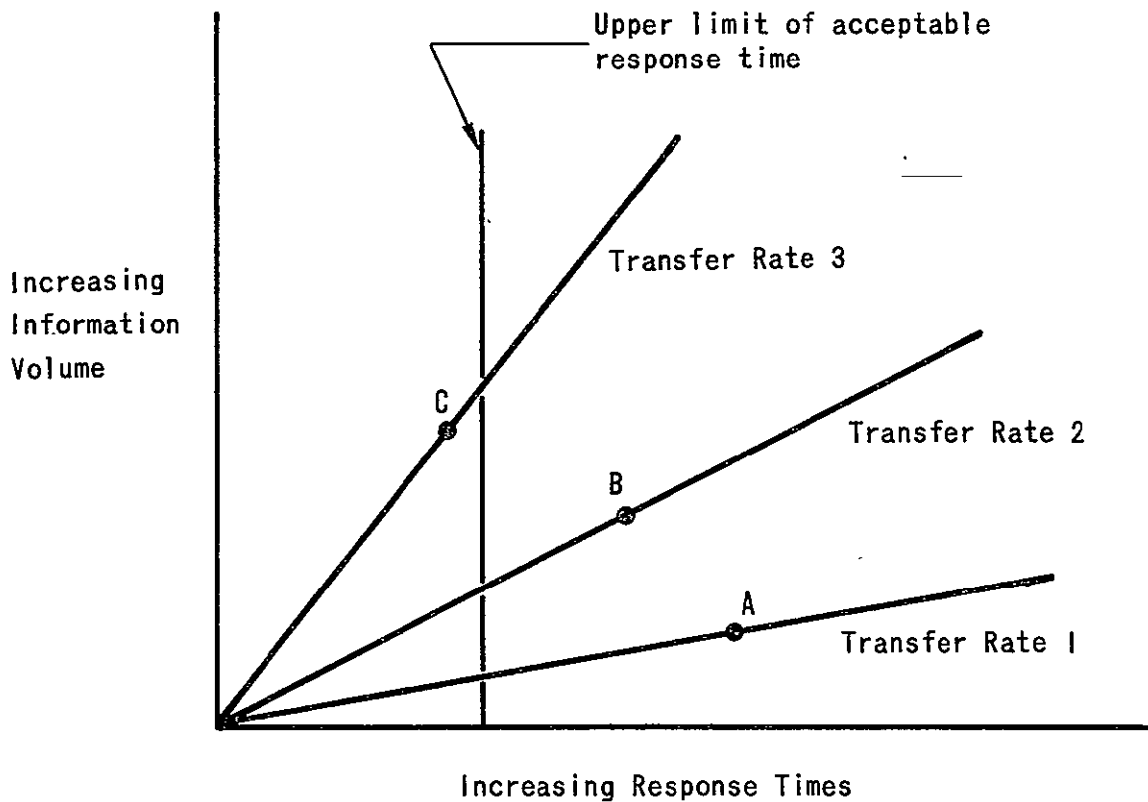


Figure 1.1 Effect of Information Volume Increase

Response time is dependent upon the device used. For example, if it were necessary to transfer a thousand order ten percent populated flexibility matrix by letter or report using a human typist, it would require between fifteen and thirty hours at a steady typing speed of fifty to one hundred words per minute discounting errors. Let this be response time A at rate 1 on figure 1.1. A more efficient method could be punched cards. If each card holds ten words and the card punch rate is one hundred cards per minute, the transfer time will be one and two thirds hours, shown as point B at rate 2 on figure 1.1. If magnetic tape or disc is used, the response time would be one to ten seconds shown as rate 3 on figure 1.1. Hence, acceptable response times are volume and device dependent.

Deteriorating Reliability - As the volume of information increases the ability of humans to maintain reliability decreases. Hence, a capability must be sought that will provide nearly perfect reliability and still have the transfer rate necessary to produce the required response times.

Diminishing Control - An organization is managed by a small number of individuals. Data is generated and used by a large number of individuals. Control of creation and changing of the data base is dependent upon the ability to collect and contain the data in a manageable form. Hence a capability is required to store the data base and provide control methods.

Obscuring of Information - The thousand order flexibility matrix in the previous example, coupled with a load matrix, contains the deflections for a thousand points, but it does not communicate those deflections to a user unless acted upon in some way. Hence, for high volumes, methods are necessary to manipulate, extract, and display the precise information needed by the user to make a decision.

The problem is aggravated and compounded when several disassociated groups become involved such as two or more companies or a company and a government agency. In these instances, local jargon and definitions, local methodology, local data formatting and local preferences become part of the problem.

Boeing's IPAD system design exploits the capacity of the computer to process, transmit, and store data rapidly and reliably to augment man's ability to communicate.

2.0 ANSWERS TO TASK 2 QUESTIONS 1, 2, 3, 12, AND 13

Answers to Task 2 questions which relate to the IPAD system design are presented in this section. The remaining Task 2 questions which relate to the support of the design process are answered in volume II and those which relate to the user requirements are given in volume III.

Task 2, Question 1 -- How should the (IPAD) system be organized to provide sufficient flexibility to accommodate independently developed codes, pre-existing and/or those created in the future?

The system organization should be able to accommodate multiple language processors, either compilers or translators and provide a mechanism for data structure transformations.

IPAD should accept other language processors to either directly compile to object code for a native mode version of the application code, or to provide language converters for interfacing with major existing languages. The burden for development of these converters would be decided on a case by case basis.

The stored data definition is the mechanism for interfacing inconsistent data structures. The user must supply such a definition for each data structure type and logic must be provided to convert from one to the other. When this is done for a particular convention, all other sets of pre-existing code using the same conventions may then enter the system without additional effort to define the data structure conventions.

Task 2, Question 2 -- What computer languages will be admissible in the pre-existing codes?

There will be a standard OM language for IPAD (see Volume VI). Additionally, any language that is acceptable to the host operating system is acceptable to IPAD, although the user may have interfacing problems between codes of different languages. IPAD will execute any code compiled on the host system, but cannot automatically interface data between codes having different input/output conventions (see question 1).

IPAD is not dedicated to working with one language in its library of coding modules. Since IPAD is using the host operating system for as many utilities as possible, any compiler that can be called as a system utility is acceptable. The consequences of the use of arbitrary languages are:

- o Input/output data structures may not interface with current IPAD data.
- o Any or all of the specialized IPAD features may never be usable.
- o An unknown quantity of machine dependence may be introduced.

Task 2, Question 3 -- What degree of machine independence is acceptable to IPAD?

Machine dependent code should be restricted to those areas concerned with the host system interface. No portion of the IPAD system communicating directly with the user should be machine dependent; i.e., the user interface logic should be independent of the host system. Machine dependent code for efficiency purposes should be done only after the performance of machine independent code is clearly demonstrated to be unacceptable.

Task 2, Question 12 -- What will be the impact of the next generation computers on IPAD?

Quantitatively the question is not answerable at this time. Qualitatively the following areas could be affected:

- o increasing size and reliability of the data base,
- o introduction of new source language capability matching new hardware logic,
- o larger number of simultaneous users possible,
- o greater involvement in multi-machine networks .

The primary aspects of fourth generation computers which could affect IPAD are:

- o array type arithmetic,
- o virtual memory,
- o. distributed computing logic,
- o significantly faster CPU operations,
- o larger auxiliary storage devices.

Some of these are direct benefits and some will require IPAD system modifications and internal redesign in order to receive significant benefits.

Task 2, Question 13 -- What is the first release capability for IPAD which should be developed for subsequent extension.

Specific capabilities for three phases of IPAD development are given in table 2.1.

Table 2.1 is related to the design nodes of section 6.2. Continuity in task and time is the primary aim of the first release system and emphasizes the following features:

- o subtask and community libraries,
- o continuity of the user's activities through the subtask concept, and
- o constructing and executing jobs.

Table 2.1 IPAD Development Recommendations

NODE	FUNCTION AND KEY FEATURES	IPAD DEVELOPMENT PHASE		
		1	2	3
E	Subtask Set Up <ul style="list-style-type: none"> • Connection to project plans • Initializing of STL • Recovery of STL with executing STS 	None Partial Partial	Partial Partial Partial	Full Full Full
F	Subtask Command Mode <ul style="list-style-type: none"> • Command decoding • Utility set up and calling • STS interruption/restart 	Full Full Full		
T	Subtask Step Controlled Abort <ul style="list-style-type: none"> • File clean up • STS termination 	Partial Full	Full	
U	Subtask Interruption <ul style="list-style-type: none"> • STL preparation for recovery • Execution after sign off 	Full Partial	Partial	Full
V	Subtask Termination <ul style="list-style-type: none"> • Tie into plans • Tie into report • File disposition • Keeping ST records 	None None Partial Partial	Partial Partial Full Partial	Full Full Full
G	Learning About IPAD <ul style="list-style-type: none"> • Teaching mode • Automatic tie-in to each command 	Partial Partial	Full Full	
H	Searching Through the Libraries <ul style="list-style-type: none"> • Display capabilities for all LE • Selection criteria 	Partial Partial	Partial Full	Full
I	Creating Library Entries <ul style="list-style-type: none"> • Entering code • Entering data • Convenient transformations • Implicit I/O 	Full Full None Partial	Partial Partial	Full Full

Table 2.1 IPAD Development Recommendations (Cont'd)

NODE	FUNCTION AND KEY FEATURES	IPAD DEVELOPMENT PHASE		
		1	2	3
K	Modifying Library Entries <ul style="list-style-type: none"> • Editing logic for code • Editing logic for data • Display capability 	Partial	Partial	Full
		Partial	Partial	Full
		Partial	Partial	Full
M	Constructing a Job <ul style="list-style-type: none"> • OM specifications • OM control program • Job network specifications • Library variable testing • Job setup testing 	Full		
		None	Partial	Full
		Partial	Partial	Full
		None	Partial	Full
		Partial	Partial	Full
N	Executing a Job <ul style="list-style-type: none"> • Qualifier specifications • Execution time options • Execution records 	Full		
		None	Partial	Full
		Partial	Partial	Full
O	Communicating with a Job <ul style="list-style-type: none"> • Specialized input functions • Specialized output functions 	None	Partial	Full
		None	Partial	Full
P	Displaying Results <ul style="list-style-type: none"> • Selection criteria • Display capability 	Partial	Full	
		Partial	Partial	Full
Q	Disposition of LE <ul style="list-style-type: none"> • Options for outside IPAD • STL to CL • STL,CL to offline archive • STL,CL to offline print 	Partial	Partial	Full
		Full		
		None	Partial	Full
		None	Partial	Full
W	Defining LE or LV <ul style="list-style-type: none"> • Redundancy checking 	Full		
GA,HA,IA, KA,MA,NA, OA,PA,QA, WA	Interrupted States <ul style="list-style-type: none"> • Anytime interrupt • Interrupt at pre-selected states 	Full		
		Full		
General Features	Privacy/security in CL	Partial	Partial	Full
	Redundancy checking on name references	Full		
	Recoverability of IPAD relative to HOST	Partial	Partial	Full
	Access/permission code checking	Partial	Partial	Full
	Support for Report and Plan	None	Partial	Full
	Interactive Graphics Support	None	Partial	Full

3.0 DESIGN REQUIREMENTS

The user requirements are the driving consideration in the IPAD design. The IPAD system that is implemented will be a balance of user requirements against software and hardware constraints to achieve an improvement in cost, timeliness, and/or technical capability over methods currently in use for product design.

The basic user requirements for the IPAD system are:

- a) Continuity over task and time
- b) User interface
- c) Privacy, security, control, integrity
- d) Reliability

The dominating requirements are a) and c). Taken in their broadest sense they imply

- a) a system that supports direct communication of technical information between organizational entities;
- b) a system that accepts as a single task, work involving many users that runs over time periods of days, weeks and months;
- c) a system that supports all of the computational, data storage, data display, data management, and data communication requirements of an entire organization engaged in the development of a product or products, and;
- d) design control both through the automatic data management and integrity controls built into the system and through controls made directly available to the management of the organization.

In this section the user requirements, independent of current software and hardware constraints, will be described.

3.1 CONTINUITY OVER TASK AND TIME

The design process flow charts developed in Volume II are representative of the type and organization of tasks necessary to design an air vehicle. However, they are only representative. The process actually followed will be an outgrowth of the product being designed, the organizations involved and the preferences of individuals at every level. Hence, a computer system designed to only perform the tasks and sequences shown in Volume II would have short term value to some parts of the aerospace industry and very limited value to the industry as a whole. To overcome this limitation, a study was made of the general design environment. It was found that continuity of activity and data over task and over time was an essential characteristic of the design environment. Continuity over task and time affects the design process in the following ways:

- a) Organizational Hierarchy
- b) Integration of Individual Contributions
- c) Phasing of Design Levels

Organizational Hierarchy - There is a hierarchy of planning and control associated with the development of a product. Information flows continuously through the hierarchy as shown in figure 3.1. The terms in the parenthesis are basic-descriptors of the primary interest at each level. While the labels of company, product, etc. are somewhat arbitrary, there are several characteristics that seem universal.

- a) There is a level at which real work on the product design is accomplished. Above that level, work is centered around planning and management control. Below that level, work is centered around preparation of tools and methods. In the hierarchy shown in figure 3.1, the level of real work on the product is at the subtask level.
- b) Each level tends to transmit information above and desire action from below.
- c) Those above tend to be interested in what is being done; those below tend to be interested in how things are done; while the user concentrates on the actual

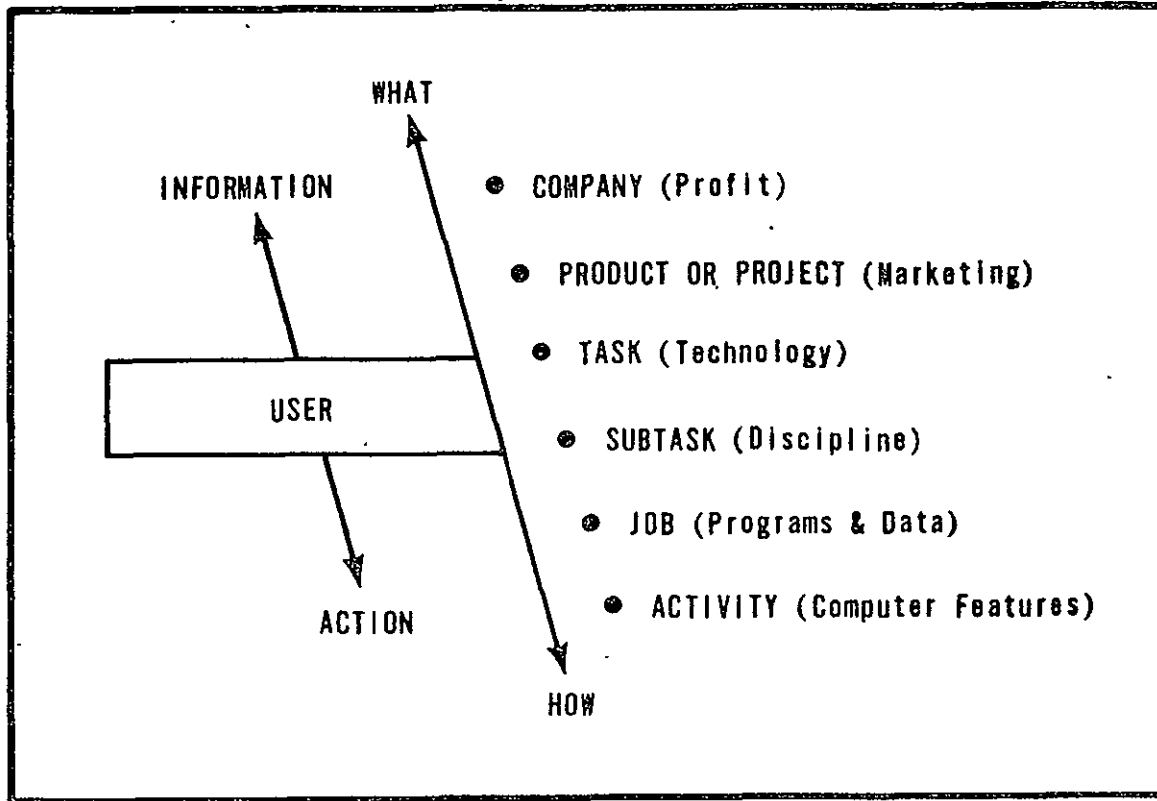


Figure 3.1 Organizational Hierarchy of Product Design

work, varying his interest between what and how depending on the immediate situation.

- d) The number of levels is not uniquely six, but it is neither large nor small compared to six.

Integration of Individual Contributions - A user of IPAD will execute a job, several jobs, or the same job several times in order to complete a subtask. The same user and other users will complete other subtasks, which, together, will form a task. Many tasks may be required to complete a project (which may be a product). Figure 3.2 illustrates this relationship. Projects, tasks, subtasks, and jobs may be large, small, or nonexistent depending on the circumstances. Some possible

examples are given in figure 3.3. On large projects, the number of users may be many hundreds and the volume of data may be of the order of billions of words. Each individual working on the project both receives and contributes data and information. The effectiveness of each individual contribution depends upon the effectiveness of his ability to communicate.

Phasing of Design Levels - In the studies performed in Volume II several levels, or phases, to the design process were defined. A different design function is performed at each level. Each level has its own characteristics of time, data volume, technology required, etc. These levels will typically be time phased as shown in figure 3.4. In general, each succeeding level represents a refinement of the product design. Essential information in the form of data and conclusions is passed between the levels, as necessary, to ensure continuity of the design process.

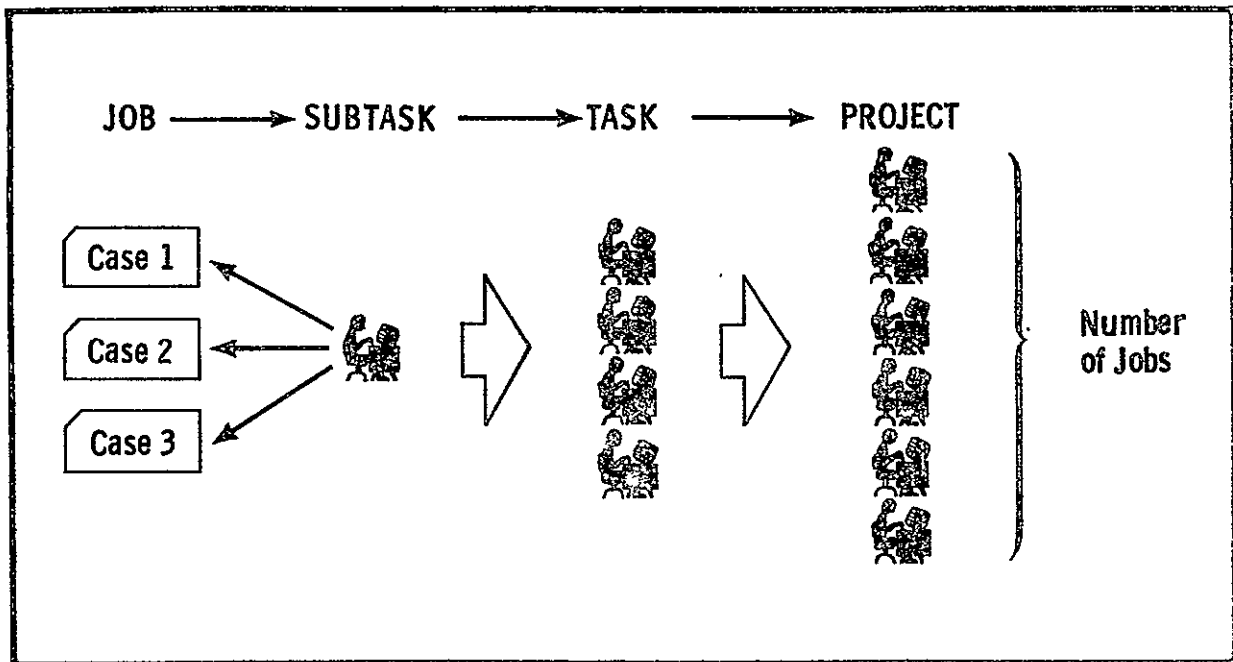


Figure 3.2 Project Growth

<u>PROJECT</u>	FIND BEST SST	FIND BEST DELTA-WING SST	FIND USES FOR S.A.S. ON AN SST	NONE
<u>TASK</u>	FIND BEST DELTA-WING SST	CONFIGURE IN LEVEL III	USE S.A.S. TO IMPROVE RIDE QUALITY	NONE
<u>SUBTASK</u>	CONFIGURE IN LEVEL III	FIND BEST CONFIGURATION WITH SUBSONIC L.E.	CHANGE S.A.S. ELECTRONICS ONLY	DEVELOP GENERAL SWEEP THICKNESS TRENDS USING WIND TUNNEL DATA
<u>JOB</u> (Execution)	SUBSONIC L. E. 4-ENGINES ETC.	4 ENGINES ETC.	SYNTHESIZE RIDE QUALITY GAINS & FILTERS	ANALYZE ONE SET OF WIND TUNNEL DATA

Figure 3.3 Examples of Projects, Tasks, Subtasks, and Jobs

In summary, the significant characteristics of this environment are:

- a) There is continuity in the day-by-day work within the organizational hierarchy, between individual contributors and between the several levels of refinement of the product design.
- b) There is a flow of information (data, directives, criteria, conclusions, etc.) throughout the entire product design community. This flow of information attempts to associate plans made, work done, and tools and methods used into a single congruent whole.

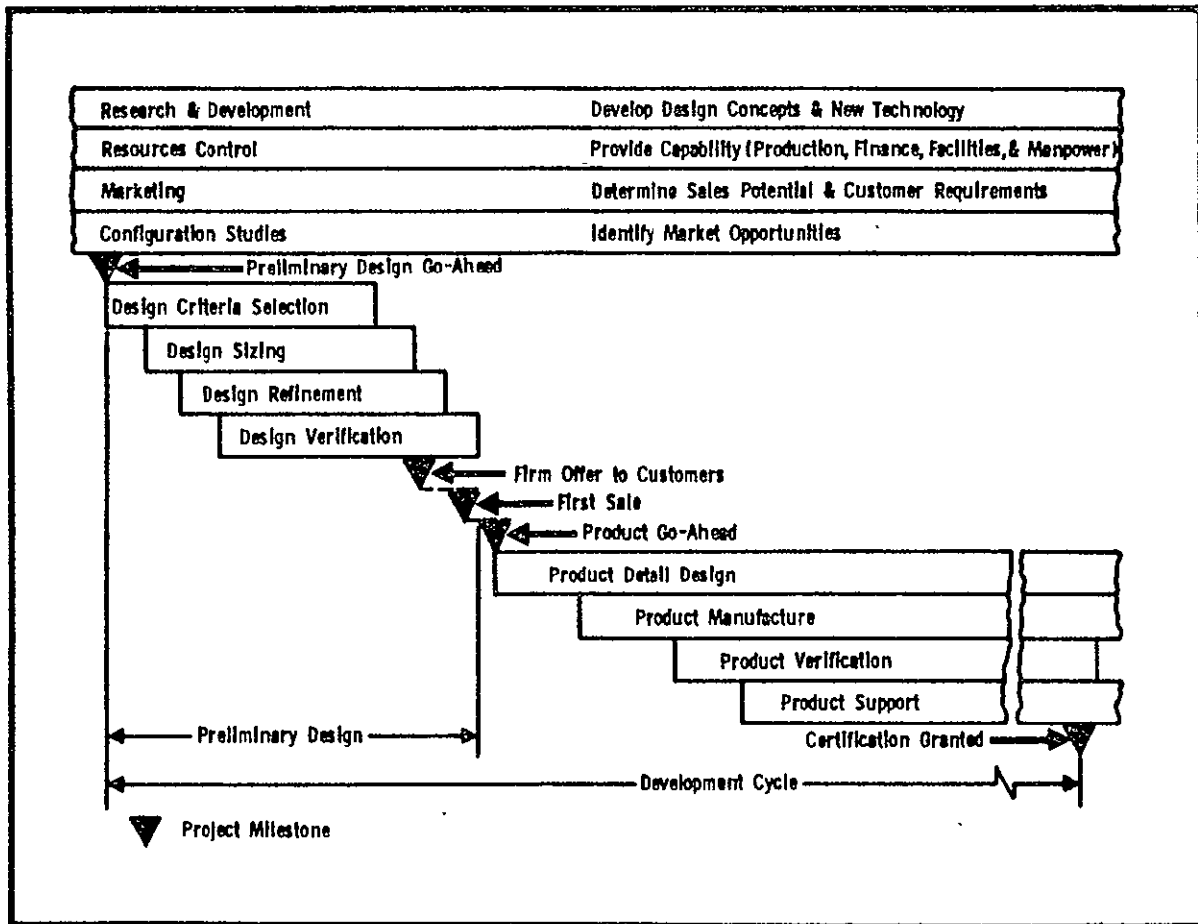


Figure 3.4 Phasing of Design Levels

- c) At every level of the organization and of the design there are individuals doing the actual work. Those above give direction, review results and exercise control. Those below prepare tools and methods and supply information.
- d) Design activities are typically ongoing over periods extending into years and decades.

It is a user requirement that the IPAD design be compatible with and provide direct software and hardware support to this environment.

3.2 USER INTERFACE

Although the design process flow charts in Volume II represent the procedure a typical design organization might follow, they do not represent the general activities users perform. An understanding of the general activities a user performs is necessary to design the IPAD system. To help gain this understanding, a problem solving model was developed as shown in figure 3.5 and given in detail in Appendix B. This model was useful in isolating particular capabilities so that the system design could be modularized effectively and general language statements could be developed.

IPAD is primarily a design tool. Hence, its basic organization is for a human hands-on operation using a command structure that makes the computing system essentially transparent to the user. As a consequence, the user is placed in a design environment entirely compatible with his own behavioural characteristics and the characteristics of the task he is performing. The principal features of this environment are given below.

- a) Accessing - Accessing will be through a personal terminal, i.e., a terminal associated with one user at a time during a work session. The minimal personal terminal will be an alphanumeric CRT with passive, low resolution graphics. High resolution interactive graphics, hardcopiers, remote job entry devices, and other equipment will vary from installation to installation.
- b) Capabilities - The IPAD system will provide the following capabilities:
 - 1) DEFINE - Entering or modifying definitions in the IPAD libraries. The definitions include abstracts, variable names, correspondence tables, and other information necessary to interface an element with the data base and provide descriptive information to the user. An element is a data set, a module of application code, a display format, etc. DEFINE is separate from ENTER to allow, for example, predefinition of variables pertaining to a data set by a single focal point followed by the actual entry of data by other persons who must then conform to the predefinition.

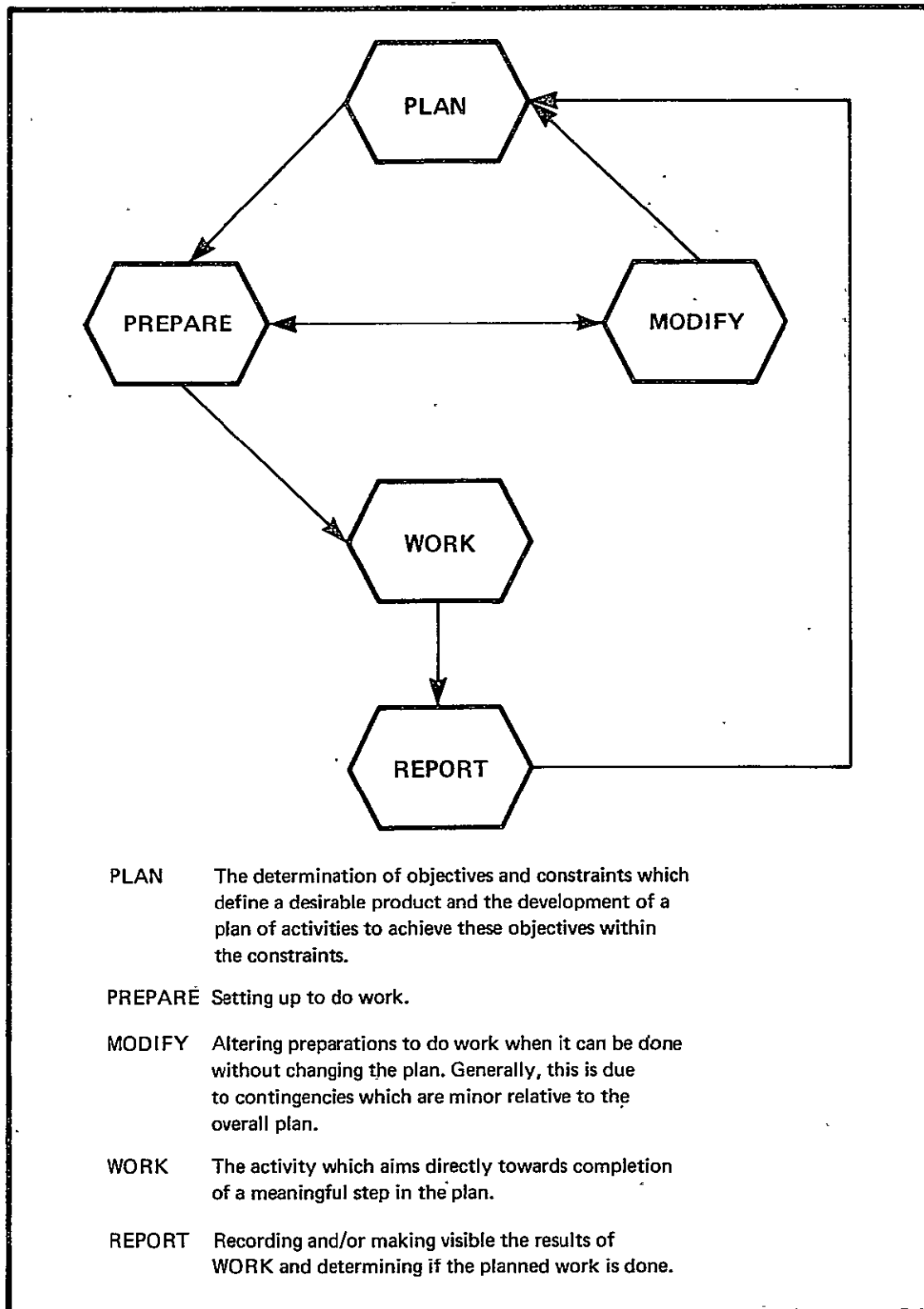


Figure 3.5 General Work Flow

- 2) ENTER - Entering the actual library element. This permits the user to enter information which conforms with the definition made using DEFINE. Many sets of information may be entered following a single DEFINE, but each will have qualifiers supplied by the user and the system to fully identify and distinguish them.
- 3) TRANSFER - Moving elements between libraries within IPAD. This allows movement of elements between private and community libraries. It does not allow movement of information to a location remote to the IPAD installation.
- 4) SEND - Sending library elements to a location remote to the IPAD installation.
- 5) EDIT - Locating and modifying existing library elements. Automatic version changes will be made by the system to accurately trace antecedents and preserve integrity for other users.
- 6) PURGE - Erasing entire library elements. System controls will exist to ensure against purging of elements still being used or saved by some segment of the user community.
- 7) COMPARE - Making comparisons between sets of information or between information within the system and information supplied by the user. This would allow, for example, a check of all moduli of elasticity within a data set to ensure they are within a given range.
- 8) CONSTRUCT - This triggers a dialogue mode in which the user may form executable code from groupings or modules of code previously entered into the system.
- 9) EXECUTE - This causes a particular set of code formed through CONSTRUCT to be executed. If the executing code has a language structure of its own, the user will interact in that language with complete transparency of the IPAD system. Commands will be available to interrupt an executing code set to review its progress, change its direction, or discontinue execution.

- 10) DISPLAY - Bringing information from an IPAD library to a display device. Display formats may be entered separately through a DEFINE and ENTER. There will be many display formats, each providing a particular class of display. Generally, activation of the display will trigger dialogue to guide the user in inputting necessary parameters.
- 11) FIND - Locating information within the libraries. This provides two capabilities: (a) locating particular sets of information such as technical code or data sets, and (b) locating particular items within a data set.
- 12) MESSAGE - Sending messages through the IPAD system to another user.
- 13) LEARN - A tutorial state that provides a programmed learning course in the use of IPAD.

In addition to the specific capabilities defined above, the system will also provide for (a) short term pauses and returns allowing execution of other capabilities between the pause and the return, (b) long term interruptions extending over log-offs and log-ons, (c) help from the system to the user to prompt him, tell him of missing information, explain commands which he has obviously misunderstood, and otherwise smooth and assist the execution of work.

- c) Communication - The transfer of data between application modules or between application modules and the system libraries will be transparent to the user. The actual locations of stored information will not be known to the user. Rather, movement or communication of information will be accomplished by the user through command statements utilizing generic names and adjectives.
- d) Human Factors - IPAD is a "hands-on" design oriented system. Hence, the characteristics of the user are an important consideration and include:
 - 1) the characteristics of the human mind,
 - 2) the experience or state of proficiency of the user, and

- 3) the characteristics of the task being performed.

3.3 PRIVACY, SECURITY, CONTROL, INTEGRITY

A review of the design process flow charts in Volume II and the manner in which an organization would proceed to work these projects indicate requirements for privacy, security, control, and integrity of system code, application code, and data. These requirements are as follows:

- a) There is a need for both private and public data regions. Private data regions provide space where an individual user can do scratch work, correct errors, etc. Public data regions provide space where data important to many users can be stored, accessed, and controlled.
- b) There is a need to protect information against unauthorized access.
- c) There is a need to control use of the system and of the data base.

3.4 RELIABILITY

The reliability of the IPAD system, hardware, and operating system should be such that system unreliability need not be a specific planning consideration for IPAD users. No definitive studies have been made to establish the precise parameters and ranges within which this criteria is satisfied. Nevertheless, it is an essential criteria.

The reliability of application modules is outside the control of the IPAD system. However, standards should be established and a rating system developed and implemented whereby application modules can be classified according to established levels of reliability.

4.0 DEFINITIONS AND ABBREVIATIONS

Activity Record (AR)	- Part of a subtask library entry, setup by IPAD for use in subtask communication, documentation, recovery, and accounting.
Coding Module (CM)	- A specific collection of symbolic code that contributes to the definition of one or more operational modules.
Community Library (CL)	- The set of all programs, data, and reference information available to the total community of IPAD users at any given installation.
Data Manager System (DMS)	- The collection of software responsible for information flow into and out of the IPAD data base.
Explicit Input/Output	- Input/output action to/from a library entry which is under the control of a user program. The data manager is responsible only for the library entry as a unit, and, in general, is not capable of interpreting the contents of any library entry handled in this way.
IPAD Data Base	- The collection of all information contained in the community and subtask libraries.
IPAD Executive (IE)	- That portion of the IPAD software used to control the basic IPAD functions. It is the primary interface to the user.
Implicit Input/Output	- Input/output action which is under control of the data manager. Information transfer by the data manager is in terms of library variables.
Job (J)	- A specific sequence of executable operational modules and/or other jobs which produces meaningful results for a user.

Library Entry (LE)	- The basic unit of storage in the IPAD data base is the library entry. The LE consists of a library directory entry (LDE) and an associated library text entry (LTE).
Library Entry Dictionary (LED)	- A dictionary containing definitions of all the library entries in a library.
Library Directory (LD)	- An index to all the entries in a library.
Library Directory Entry (LDE)	- That portion of a library entry containing control and referencing information for the associated library text entry (LTE).
Library Text Entry (LTE)	- That part of a library entry containing the data associated with the entry name.
Library Variable (LV)	- An alphanumeric data item whose engineering significance is defined to IPAD.
Library Variable Dictionary (LVD)	- A dictionary containing definitions of all the library variables in a library.
Operational Module (OM)	- An executable collection of coding modules which contribute to the definition of one or more jobs.
Operating System (OS)	- The operating system for the host computer within which IPAD executes.
Personal Terminal	- The electronic or electro-mechanical device providing the primary path for the user to access IPAD data and programs.
Project (P)	- The total set of subtasks to be performed during a design or analysis effort.
Project Plan	- The definition of all project tasks,

	subtasks, and the associated control in terms of a pert-chart type network.
Project Report	- The collection of reports expected to be completed during the progress of the project.
Qualified Library Entry Name (QLEN)	- An unqualified library entry name with a qualifier attached indicating a specific instance of the LE.
Stored Data Definition (SDD)	- The specifications for a logical information structure for one or more library entries in IPAD.
Subtask (ST)	- A sequence of IPAD activities which represents a meaningful step in a project.
Subtask Library (STL)	- A library that is private to an IPAD user during the execution of one of his subtasks. Each subtask will have a single associated subtask library.
Subtask Records	- Records in each subtask library for the purpose of holding activity and other information during the life of the subtask.
Subtask Step	- A single step occurring in a subtask, normally defined by a host operating system control card or the execution of a single IPAD utility program.
Task (T)	- A subdivision of a project.
Unqualified Library Entry Name (ULEN)	- A generic or root name of a library entry. Specific instances of data may be identified by a ULEN appended with a version number and/or an additional qualifier.
User's Identification (UID)	- A unique identifier associated with each user of IPAD. It is mandatory that this ID be associated with a person and not an activity or an organization.
Version Number (V)	- An identifier appended to an unqualified library entry name to record the occurrence of a change.

5.0 FUNCTIONAL DESIGN DESCRIPTION

Early methods of organizing computing tasks involved batching work of a similar nature and relying on well trained personnel to maintain a reasonably efficient operation. As volume and complexity increased, many of these administrative functions were shifted to the computer, leading to the present day "operating" and "monitor" systems. Each new operating system development had two basic objectives:

- a) more efficient processing, and
- b) new capabilities to aid the professional programmer.

In contrast, software has not been generated which helps the applications user organize and manage his work. Moreover, most operating systems are "one run" and "one user" oriented while typical applications require numerous runs, spanning several days or months and involve many people performing inter-related activities. IPAD is designed to help the applications user manage and organize his work. It will support continuity of work on the computer involving many separate work sessions extending over long time periods and requiring communication between users through the data base.

5.1 PRIMARY SYSTEM FEATURES

Full Project Support - A project is a set of tasks and a task is a set of subtasks. Within any project the sequence of tasks and subtasks is not arbitrary. Figure 5.1 is a representation of two projects showing the breakdown of tasks and subtasks and the flow of information or interdependence between them.

There are specific items in the data base called "plans" and "reports" that support management definition and control of the work flow. Each project and subtask will be given a report skeleton at the time planning data is entered in the data base. Hence, completion of each subtask is a formally recognized event in IPAD. That is, the completion of the subtask can be recorded in a project report. If a PERT-chart type logic is used for planning, as shown in figure 5.1, the sequence of subtasks can be controlled through permission codes in the system. Managers and technical users can also interrogate the status of projects or subtasks from the project reports.

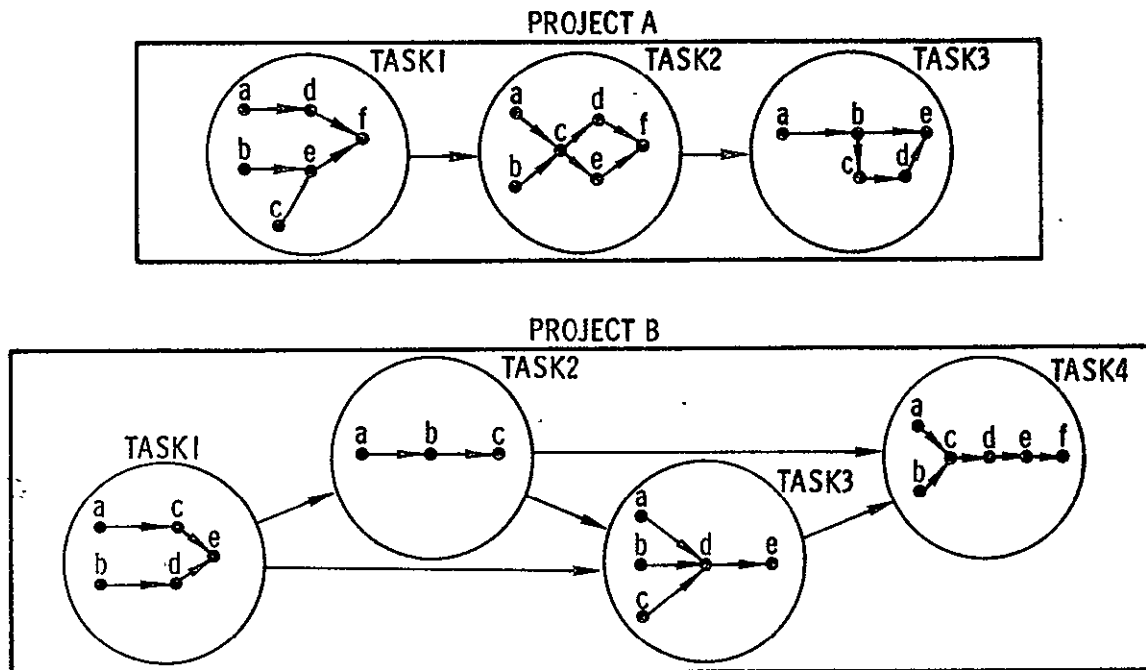


Figure 5.1 Representation of Project Plans

Continuity of Work - Subtasks are the prime working interface between IPAD and the IPAD user. Subtasks are generally associated with one user and are organized within IPAD to provide continuity of activity. All user activity is part of some subtask which was explicitly initiated by a user and must be explicitly terminated. The life of a subtask is not limited to an artificial work boundary such as a computer run or a terminal session.

The IPAD system will accept any definition of a subtask. In principal, a user may have any number of subtasks defined at any point in time, although only one would be active at a time.

A subtask is a user's private domain in which he works individually without impacting other users. He has access, through the data base libraries, to application modules, utility modules, and data common to all users. His accessing of information in the data base libraries is controlled or restricted as necessary to protect the community nature of the data base.

Continuity of activity means that from the time the user initiates a subtask until he terminates it, he works with the sense of continuity of a single session. That is, having interrupted his activities for lunch, sleep, or thinking, he will resume activity with a subtask status identical to that at the time of his interruption. This continuity will be true for either interactive or batch type work.

Continuity between subtasks (and hence between users) will be provided by the data base. As each subtask is terminated, the user will transfer entries of common value into the community library.

Library Structure - The IPAD data base is organized into a community library (CL) common to all users and subtask libraries (STL) private to each user. The characteristics of these libraries are:

- a) The community library contains all application modules, data sets, and other information available to the using community in general.
- b) Community library items may be attached, copied or transferred to the subtask library depending upon permission codes associated with the entry and the subtask.
- c) The subtask library contains all application modules, data and records associated with a subtask.
- d) When the subtask is inactive, all items in the subtask library are logically located within the community library. However, contents of the subtask library are not accessible as community library items.
- (e) A subtask library is created when a subtask is defined and continues to exist until the subtask is formally terminated.
- (f) A non-community library item in a subtask library is private to the subtask library.

The relationship between subtask and community libraries is illustrated in figure 5.2.

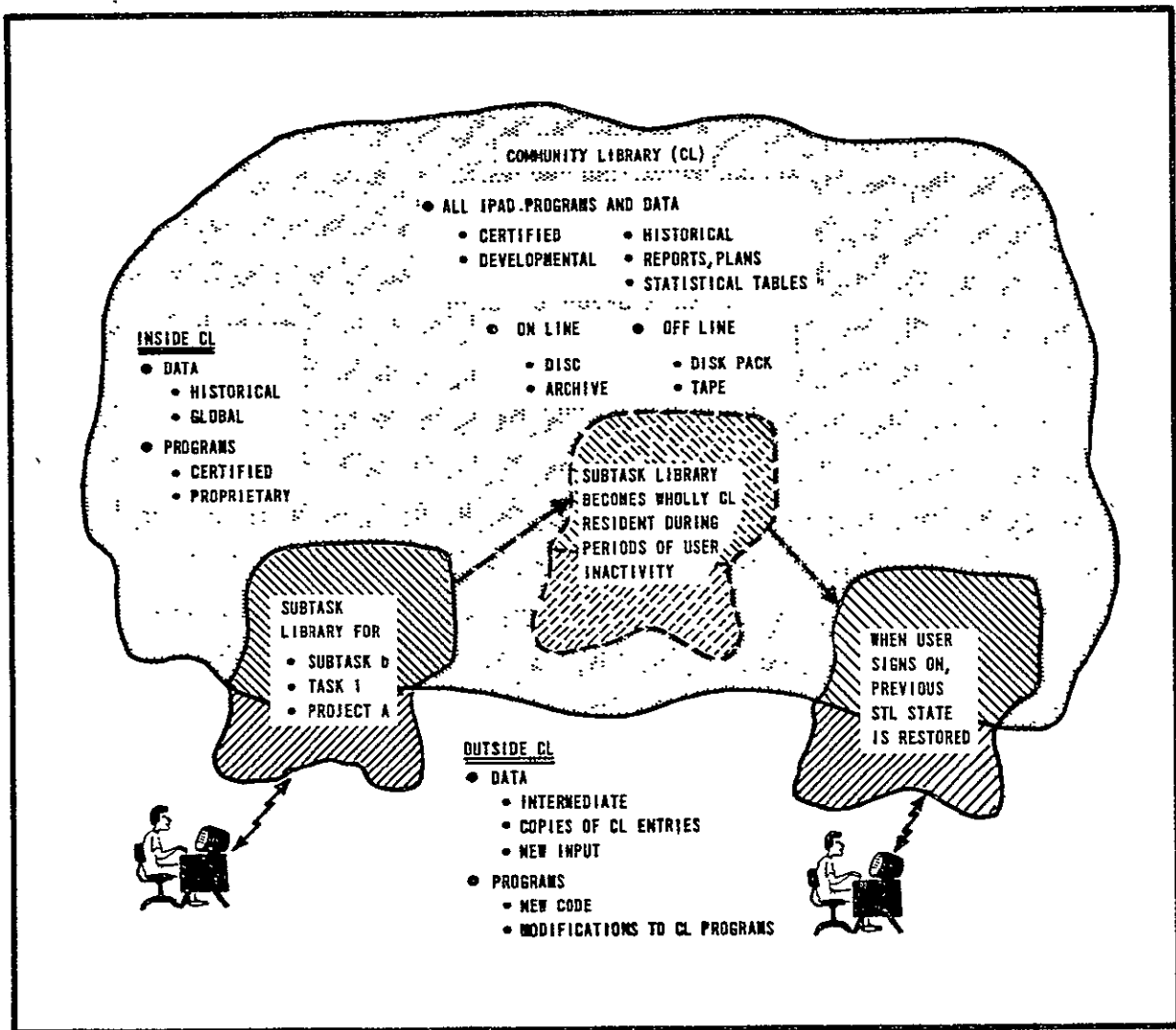


Figure 5.2 Characteristics of the IPAD Community and Subtask Libraries

5.2 WORK RELATIONSHIPS

Figure 5.3 shows a schematic relationship of work within IPAD. A project consists of

- a) Project Plans
 - o Overall Project Plan
 - o Individual Subtask Plans
- b) Project Reports
 - o Project Summary Report
 - o Individual Subtask Reports

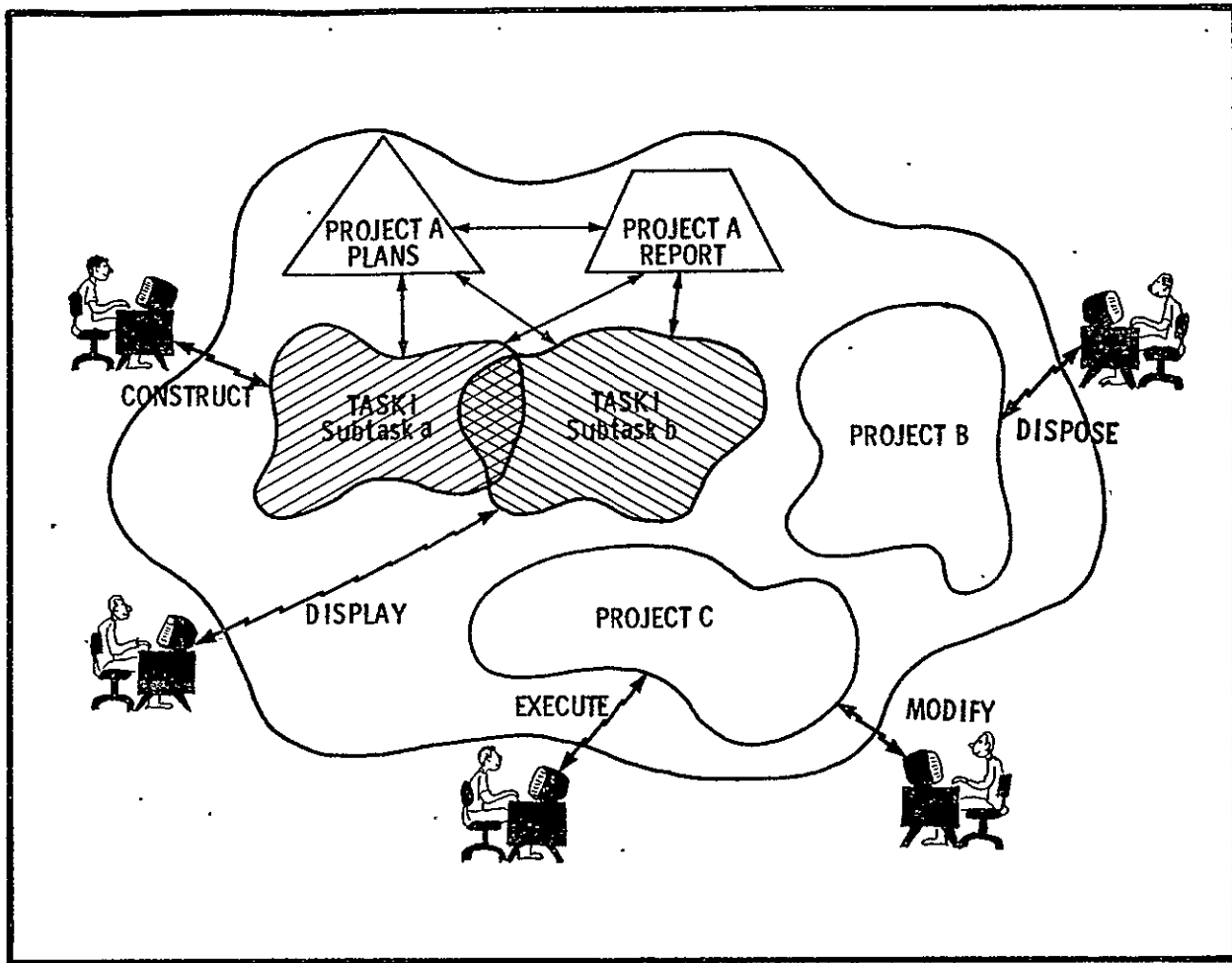


Figure 5.3 Data Base and Work Relationships in IPAD

c) Subtasks

- o All user activity in IPAD occurs in a subtask.
- o The mode of communication in a subtask is primarily interactive.
- o Batch mode is available. Continuity is retained in the subtask whether accessing is interactive or batch.
- o A user may have an arbitrary number of subtasks defined at any one time.
- o A subtask has four distinct states: defined, active, inactive, or terminated.

- o Records of subtask progress are contained in the subtask library and are used to formulate project reports.
- o Communication between subtasks is through the community library.
- o Protocol is defined to maintain integrity of data in the community library.

IPAD will provide a framework within which control and reporting requirements may be defined to meet the needs of the using organization. For example, project and task plans may be placed within the system in the form of subtask sequences as was shown in figure 5.1. Control can then be exercised such that subtasks can only be initiated as preceding subtasks are terminated and subtasks cannot be terminated until subtask reports are entered in the project report. However, project plans could be entered in the system with no control and reporting requirements or there could be a complete absence of such plans. The minimum requirement in IPAD will be that the subtask must be defined before it is activated so that resource accounting and reporting can be done by the system.

5.3 USER INTERFACE

5.3.1 Personal Terminal

The primary interface between the user and IPAD will be a "personal terminal". A "personal terminal" is uniquely associated with a given user while he is active. Each user will be identified to the system via his identification code. Since the IPAD system will not have terminal handling software, the user's first contact will be with the operating system. If IPAD is the only system requiring terminal support, the operating system will be a transparent message carrier.

5.3.2 Command Flow

In general, each command to IPAD will be mapped into one or more operating system commands. Control will then be given to the operating system. When the operating system has completed a set of commands, IPAD will be recalled to determine if all commands have been processed. If not, the above process will be repeated. When all commands have been processed, appropriate entries will be made in the subtask records and all library entries will be disposed of as requested by the user. An activity in progress may be interrupted and another activity initiated. Upon completion of the second activity the user may

continue the first activity. This sequence may be nested. The execution sequences for data and programs may be stored as job descriptions in the IPAD libraries. The general flow of control for IPAD and system commands is shown in figure 5.4.

While many languages may exist within the total IPAD domain, the IPAD executive itself needs a relatively limited language capability. The basic commands are modal statements, i.e., the command indicates the basic intent of the user. These commands are the means of executing all application modules, executive utility functions and data base management functions. Recommended basic commands were given in section 3.2.

5.3.3 Sign-On To IPAD

The sign-on procedure initiates a new subtask or restarts an existing one. A user must supply an identification number (UID) and security passwords. Sign-on will allow the user access to all library entries where his ID number appears in the permission code tables. If no subtask name is input, a new subtask will be assumed and IPAD will ask for a subtask plan identifier so that planning data can be examined to find the appropriate subtask name. Planning data can also be used to search for subtasks in progress.

If an existing subtask name is given there will be a library entry of that name in the community library, and the subtask library will be established using the information in the entry. Except for records involving time, the library contents for existing subtasks will be as if the user had never signed off, providing continuity across time lapses in activity.

If a new subtask is defined, the following information must be supplied by the user or through system defaults:

- a) User validation information for the particular project.
- b) Basic options such as display formats, etc.

5.3.4 Communication With IPAD

Having signed on, the user may initiate any of the IPAD basic commands in his command permission profile. The system will respond in one of three basic modes.

- a) Monologue - The user knows the command format and is capable of delivering complete commands to IPAD.

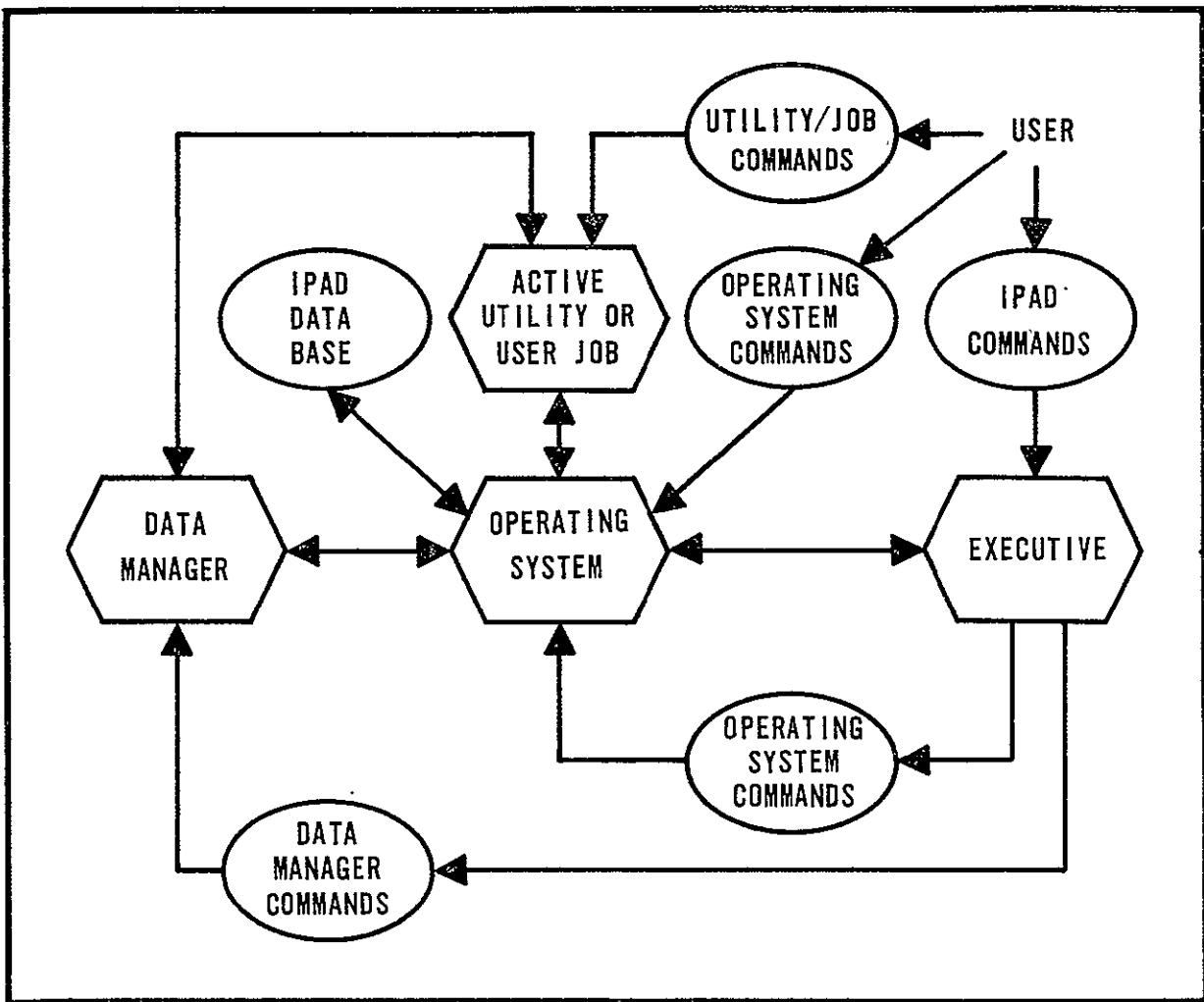


Figure 5.4 IPAD System Command Flow

- b) Dialogue - The user knows the general workings of IPAD but is not capable of entering complete commands.
- c) Teach - The user may, at any time, request assistance and IPAD will go into a partial or full teaching mode depending on the request type.

When a user's response to the system is interpreted, the following possibilities exist:

- a) Execution is possible and:
 - o The command is correct in form and intent; or,
 - o The command is correct in form but the user's intent is different from the potential execution results.

- b) Execution is not possible because:
- o IPAD understands the command but has insufficient information, or
 - o IPAD understands the basic command type, but detects an error in format, consistency, etc.; or
 - o IPAD does not understand the command.

These responses are summarized in the following table and will be built into the system. The number indicates sequence of action, and parentheses indicate optional actions.

EXECUTION OF COMMAND	RESPONSES			
	Do It	Ask If They Understand Action	Ask If They Would Like Help	Explain Error
Possible	(2) 1	(1)		
Not Possible				
- Unable		(3)	(2)	1
- Detectable Error		(2)	(3)	1
- Unrecognizable			2	1

5.3.5 Sign-Off From IPAD

Sign-off has two basic modes - inactive or terminated. If inactive, the system will save the contents of the subtask library in the community library and prevent purging or editing of those versions of community library entries associated with the subtask. If the subtask is terminated, disposition instructions for all items in the subtask library must be given. Sign-off is a discontinuation of activity on a particular subtask and may or may not terminate the activity period.

5.3.6 Batch Access

IPAD commands are available in batch mode and must be submitted in monologue fashion. Sign-on and sign-off will be similar to interactive mode, including the subtask concept so that continuity is preserved. The end of a batch run is similar to sign-off with subtask inactive. A user may submit a job through batch and, regardless of the results, make his next access either from batch or from the terminal. Thus, the IPAD system will leave an interrupted subtask in the same state for batch as for interactive access.

5.4 LIBRARIES

When active on any given subtask, an IPAD user's data base is his subtask library and that portion of the community library to which he has access. Any new information must come through his subtask library or from access to additional community library information. In these two libraries, the user is concerned with library entries and variables and the dictionary which holds their definitions.

5.4.1 Library Entries (LE)

The primary unit of information storage in an IPAD library is the library entry. In order to handle the wide variety of information expected in the IPAD data base, many different types of library entries have been defined. The most significant distinction among types is between user and system entries. User entries are composed of alphanumeric information which is either input to or output from some operational module in IPAD. This entry type is composed of one or more library variables (see section 5.4.2). A system type does not contain library variables. These entries contain source code, binary code, project plans, etc. The set of library entry types is expandable.

5.4.2 Library Variables (LV)

A library variable is defined in an IPAD dictionary (see section 5.4.3) in terms of its technical significance. This includes both its engineering meaning and its mathematical meaning (e.g., single real number, rectangular matrix, complex vector, etc.). The isolation of variables is a mechanism for organizing information transfer between technical code and between people. A library variable may be resident wholly within more than one library entry, but a multiple valued variable (e.g., a vector) may not be partially resident in several library entries. Any variable may have any number of values residing in separate library entries, but there will only be one definition of that variable in any one dictionary.

5.4.3 Library Dictionaries

Each library in the data base has at least one dictionary. Typically one would expect a subtask library to have only one dictionary, but there may be several dictionaries in the community library. When library entries or variables are referenced, a specific dictionary will be used to reconcile potential ambiguities. Generally the context of the reference will be sufficient to define the situation; e.g., the user's subtask library is assumed to contain any item referenced, and if it cannot be found, a specified community library dictionary will be used.

5.4.4 Library Entry Naming Conventions

The user will reference library entries by a generic name assigned by the entry originator. Some library entries (e.g., data sets and coding modules) will be modified during use and will therefore require version numbers. In addition, the source of data sets generated during the course of work will be identified. Hence, qualifiers will be appended to the entry name recording the names of data sets and application modules used to generate the data set. Therefore, library entry names will have the form

NAME.VERSION(QUALIFIER)

To access a library entry, the user, through direct command or through job control, will give enough of the library entry name to uniquely identify the entry. Less than that will cause the system to request more information.

Name - The name is any set of characters supplied by the user. When stored in the directory, the owners ID and password and the subtask identification will be appended to the name by IPAD.

Version - Version is both a user and IPAD generated item. IPAD will require a change in version number whenever modifications are made to a library entry without changing the name. The system will provide the capability of referencing the "latest version".

Qualifiers

The basic intent of qualifiers is to insure documentation of the origin of library entries. Both the user and IPAD establish qualifiers. When the user initially creates a library entry, he supplies a qualified name, except for those entry types which logically do not require qualifiers. As the entry is used for various jobs, its qualifier is used by IPAD to establish qualifiers for newly generated entries. The qualifier is associated with a library entry and not individual library variables within the library entry.

Figure 5.5 illustrates how qualifiers are generated during the execution of a job. The elements in figure 5.5 are defined as:

Job Components	X, Y, Z
Input Library Entries	A, B, C, F
Intermediate Library Entries	D, E
Output Library Entries	L, M, G.

If B is a library entry qualified by Q, it is identified as B(Q) where Q is either supplied by the user or is a qualifier generated by IPAD as a result of a previous execution. The qualifier generated for L in figure 5.5 at job assembly time will be

L(Z(D(X(A(NULL))) ,E(Y(B(NULL) ,C(NULL))),F(NULL))).

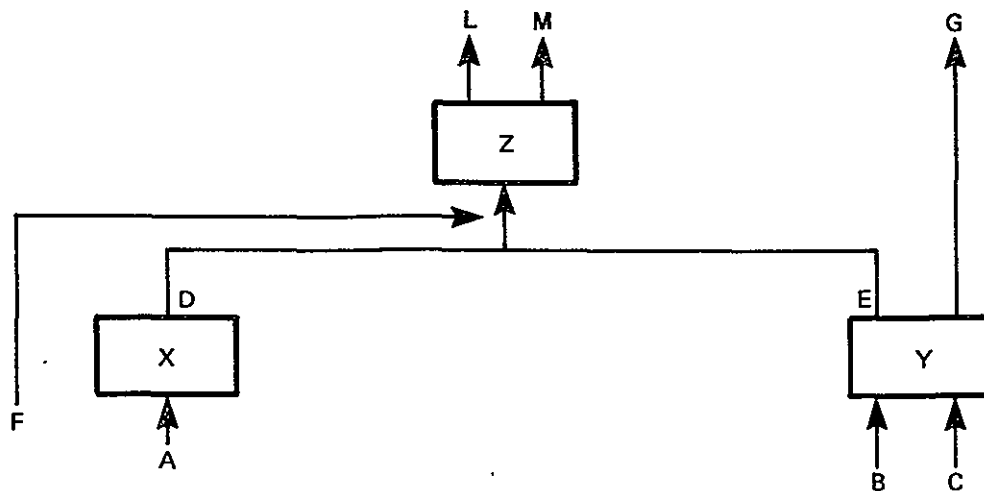


Figure 5.5 Sample Job Organization

The qualifier generated for G at job assembly time will be

G (Y(B(NULL),C(NULL))).

The user will normally not deal with the long form of the fully developed qualifier. He will, in general, use only enough of the name to insure an unambiguous reference.

The null qualifiers are supplied at execution time to complete the qualification of the output library entries for each specific job execution.

5.5 THE JOB CONCEPT

Operational modules and utility functions are executed as one or more jobs. A job is a selected set of operational modules (OM) and/or other jobs organized by a user as part of a subtask. An OM consists of a selected set of subroutines organized as coding modules (CM) to execute in a sequence as defined in a "main program". Without regard to execution sequence, the above relationships are shown in figure 5.6. Any set of source code used in several OM's should be entered as a CM to make it more visible to the user community. The same rule applies in the OM to job relationship.

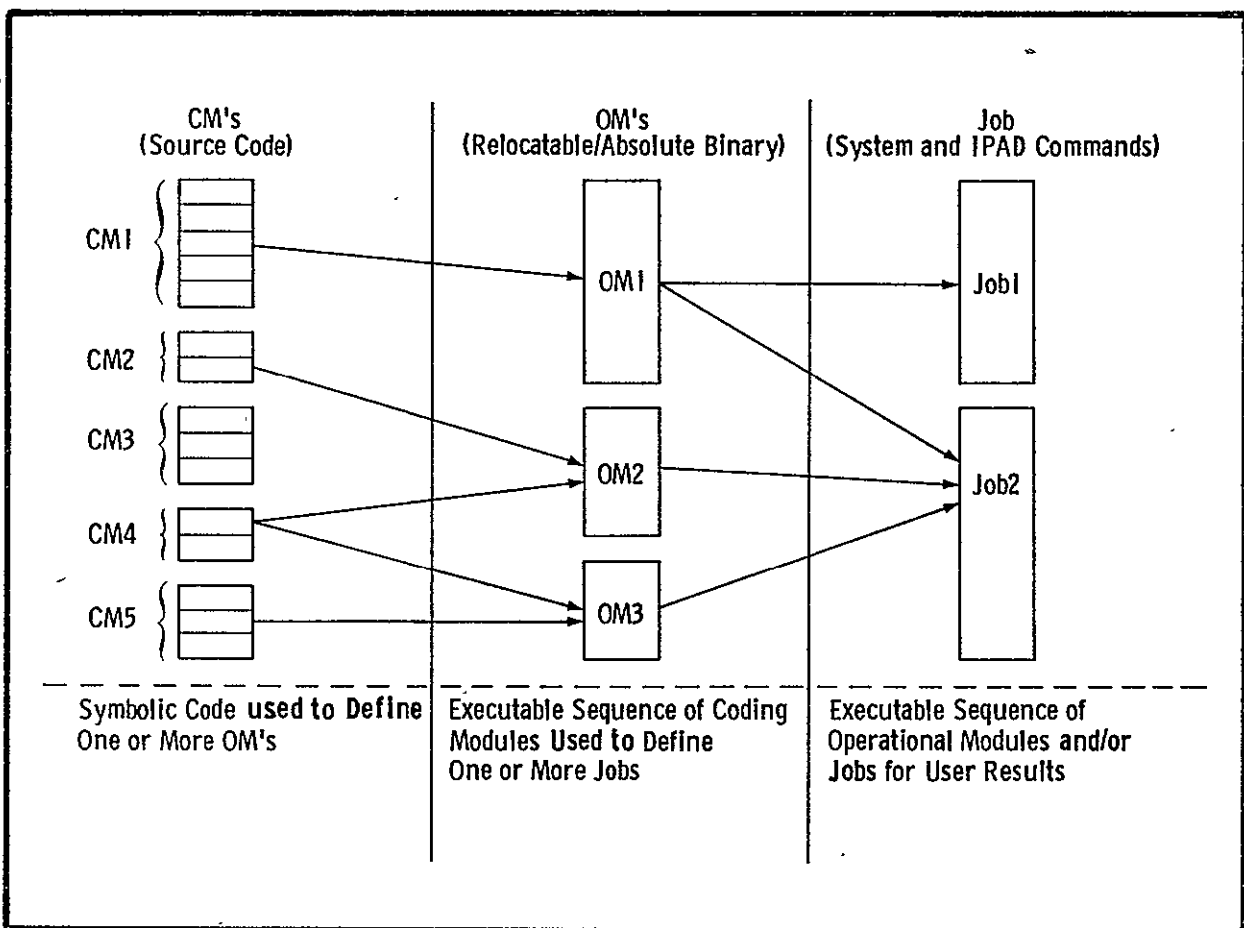


Figure 5.6 Sequence of Job Definition

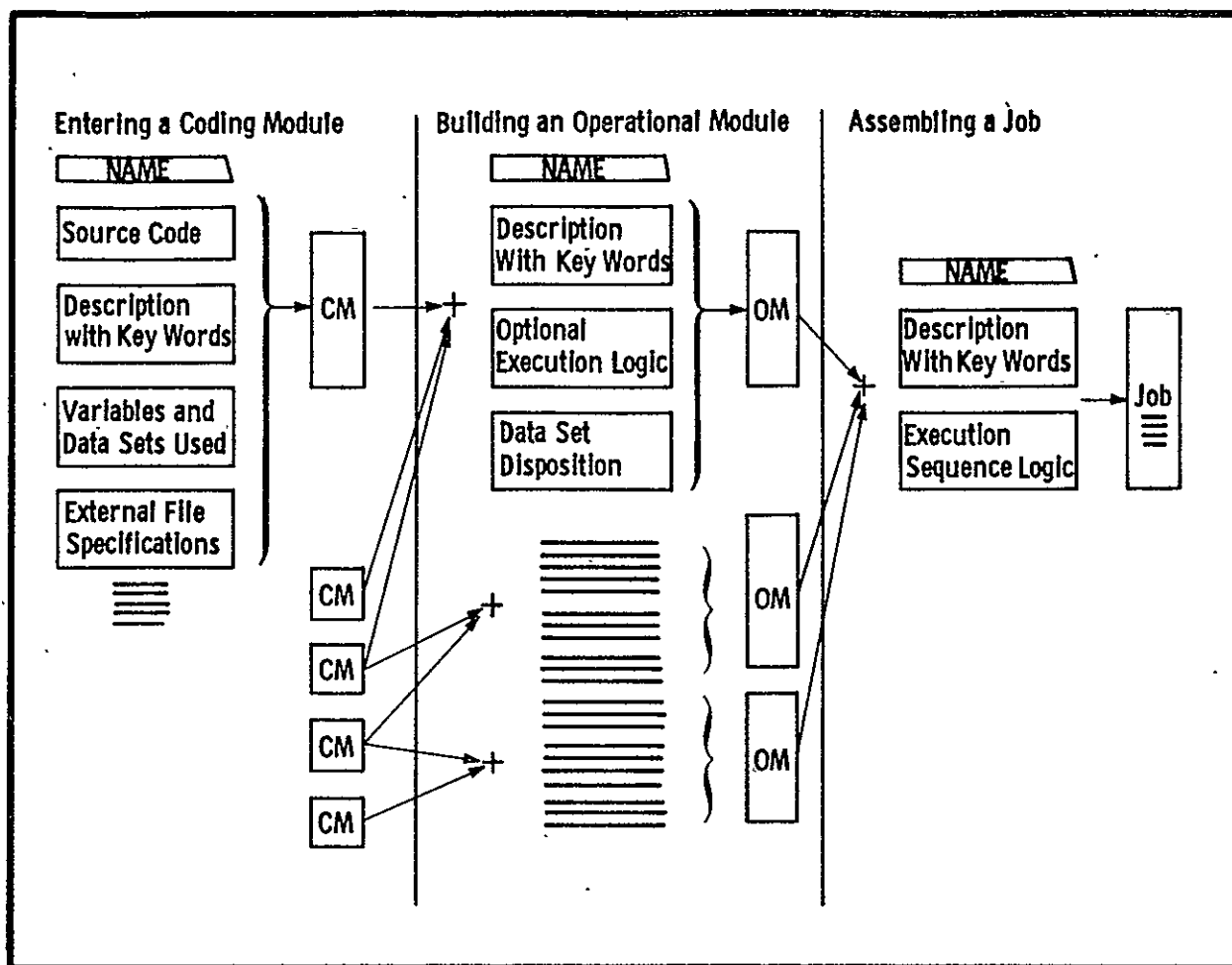


Figure 5.7 Steps in Job Construction

5.5.1 Entering Coding Modules

Source code and descriptions are entered as Coding Modules as shown on figure 5.7.

5.5.2 Building Operational Modules

A complete OM must have one main program, i.e., only one main program may exist in the CM's being packaged into an OM. If a main program does not exist in a CM, one must be written as part of the OM building process. Library entry information, as shown on figure 5.7 must be defined at OM build time. Any files used by the OM and not linked to library entries by the CM's must be linked at this time. If an OM is placed in the

community library, all its component CM's must be in the community library also.

5.5.3 Constructing Jobs

Constructing a job is similar to building an OM. An execution sequence with optional logic at OM boundaries is specified. The executable units in a job are OM's and other jobs. During job assembly the qualifiers for all output library entries will be partially constructed. Information supplied at job execution time will complete the qualifier. Job construction is illustrated in figure 5.7.

5.5.4 Execution of Jobs

When job execution is requested, the subtask library and community library are searched for the correctly qualified input library entries. The library entries for all intermediate and output library entries are set up in the subtask library and are qualified by both the set of qualifiers generated during job construction and the set received with the request for execution. All file linkages (as specified in the job definition) are set. Activity record information for this job request is written and commands are delivered to the operating system for execution.

5.6 DATA BASE MANAGEMENT

The libraries contain two categories of information structures; system defined and user defined. No formal distinction is made between these categories by the data manager, but system structures are not generally accessed directly by operational modules. The system structures are used to store operational modules, referencing information, control information, etc. User structures contain data produced and manipulated by application modules. Figure 5.8 gives a general description of the data base contents.

The user is not restricted to a fixed set of data structures. The two basic modes of access to data by application modules are:

- a) Explicit I/O - The user's code is written with detailed knowledge of the data structure of the library entry.
- b) Implicit I/O - The user's code references variables within a library entry by name, letting IPAD do all storage and retrieval.

With explicit I/O, library entries are made available at execution time by the data manager. The data manager then assumes that the user's code will handle all read/write operations.

With implicit I/O, the operational module definition contains declarations naming stored data definitions, and explicit I/O statements are replaced by commands to the data manager to fetch and store data. The data manager carries out the required I/O as specified by the appropriate stored data definition. A schematic of the two I/O modes is given in figure 5.9 with a table of comparisons in figure 5.10. An example is given in figure 5.11.

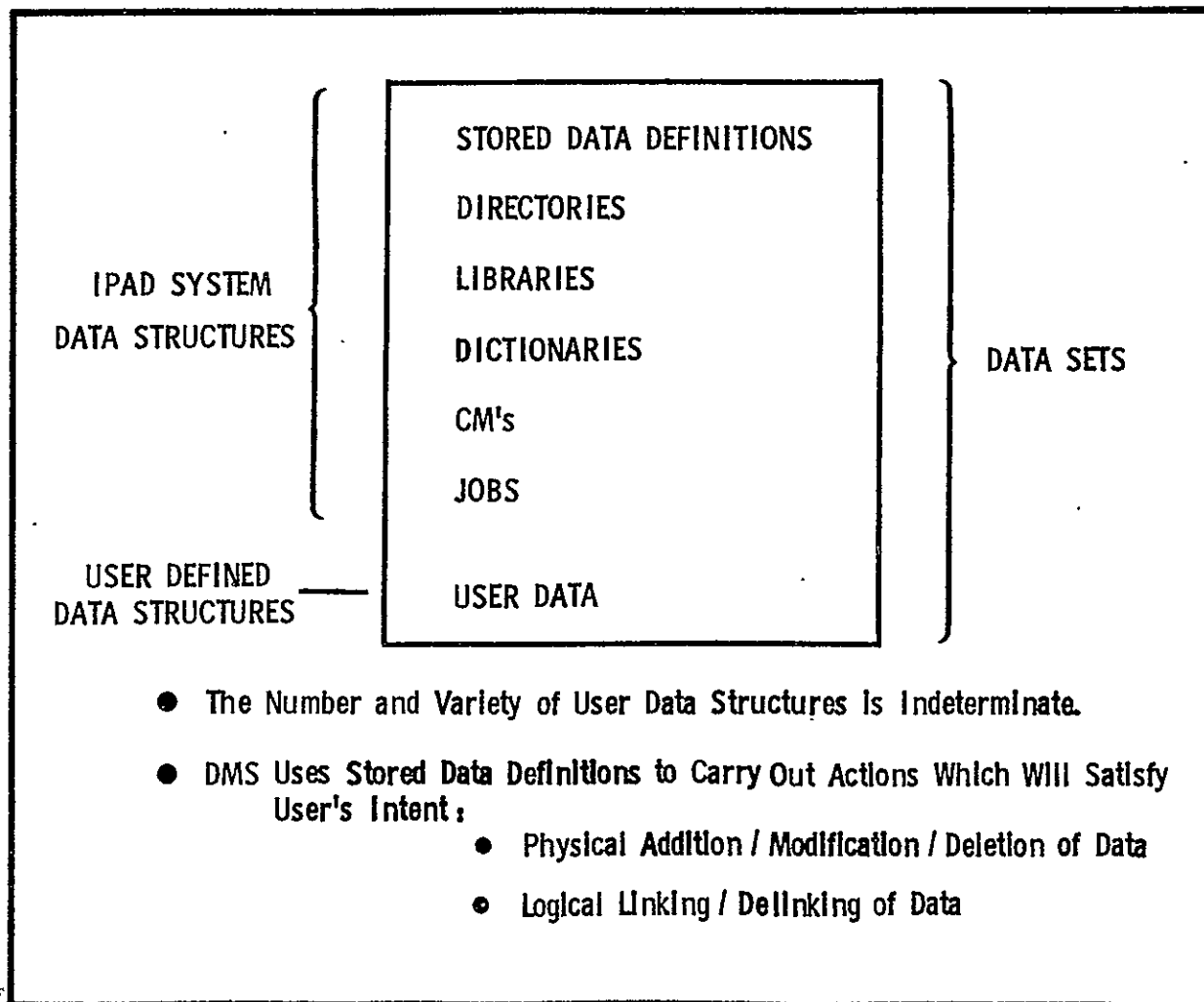


Figure 5.8 Data Base Contents

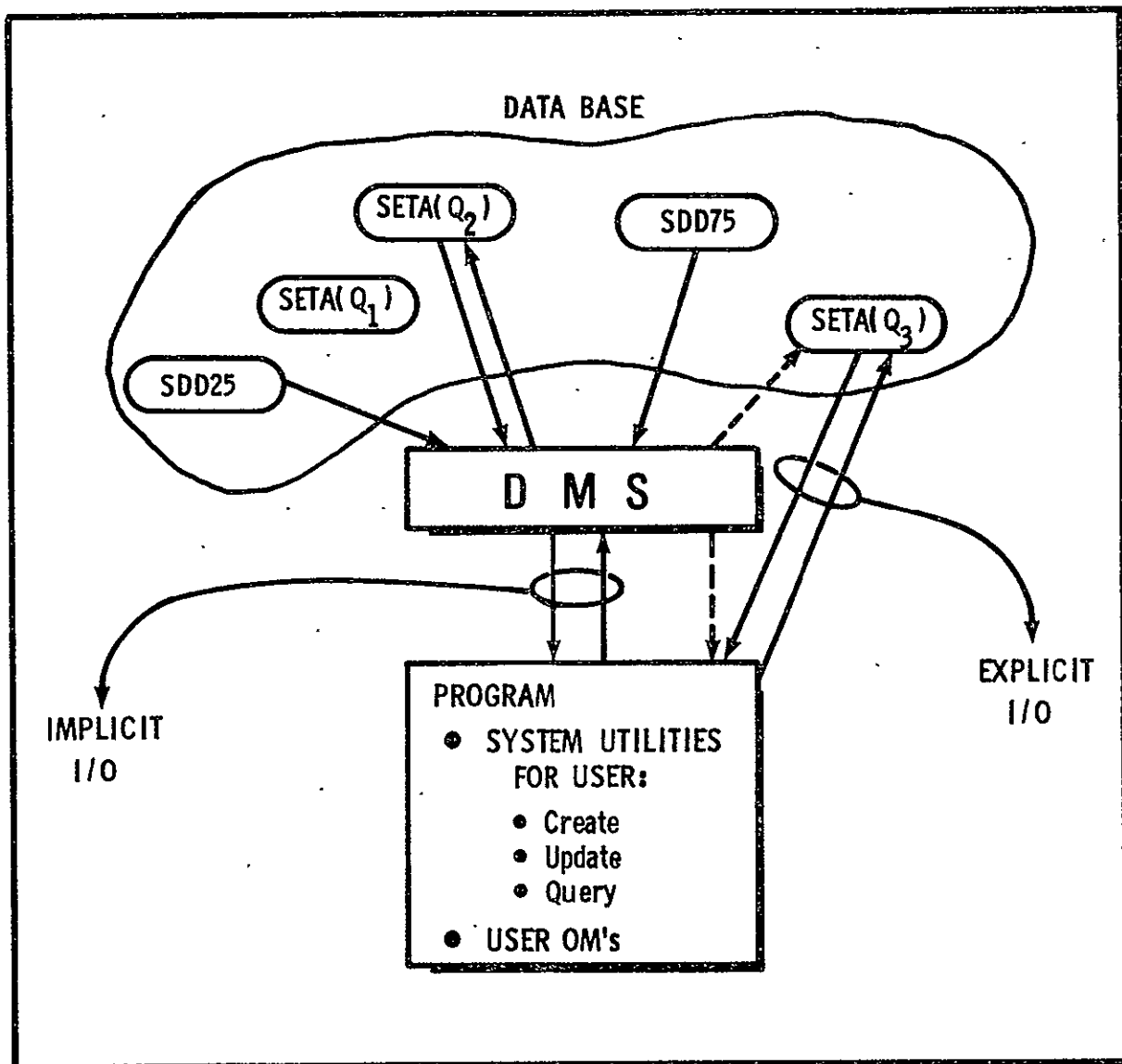


Figure 5.9 Relationships of Implicit and Explicit I/O to the Data Base

Stored Data Definition (SDD) - All information in the IPAD libraries may be accessed through a stored data definition (see figure 5.12). SDDs for user data are user supplied, avoiding the problem of forcing users to change their data formats and structures to conform to a single standard.

Each library entry in IPAD may have one or more formats associated with it. When described by a single SDD, multiple formats imply the use of subsets of the whole library entry or possibly different unit conversions. Multiple format SDDs also allow external formats (e.g. punched cards) to be described.

SDDs are mandatory when using implicit I/O and optional for library entry used in explicit I/O only.

Implicit I/O - SDDs require each variable to have a definition in the variable dictionary and a corresponding global name. The utilization of global names in the SDD eliminates many of the ambiguities that could arise when different users are interpreting data variables or when decisions are made regarding the selection of a coding module for a specific computation. The SDD also permits variables to be given local names for different subroutines. During execution, calls on the data manager will result in data being moved from the storage media to the user's working area. Data will be positioned in the user's working area so that local name references to variables in the user's code will be correct.

EXPLICIT I/O	IMPLICIT I/O
<ul style="list-style-type: none"> • DMS Connects OM to an Entire Data Set 	<ul style="list-style-type: none"> • DMS Connects User to All or Part of Specified Data Sets
<ul style="list-style-type: none"> • OM Performs all I/O Operations Without IPAD Intervention 	<ul style="list-style-type: none"> • User Requests Data <ul style="list-style-type: none"> • By Library Variable (Data Item Key) • By Position <ul style="list-style-type: none"> • Sequential Data Sets • Logical Chains
<ul style="list-style-type: none"> • DMS Functions Performed only on Entire Data Set 	<ul style="list-style-type: none"> • DMS Functions Performed on Individual Library Variables Within Data Sets

Figure 5.10 Attributes of Explicit and Implicit I/O

Pre-Existing (or Independent) Code May Require Data Input in a Certain Structure

- If Data Exists in Proper Form, DMS Connects Data to Program
- If Data Exists, But Not in Proper Form, a Pre-Processing OM Using SDD's Can be Used to Interface Pre-Existing Code Without Changes to the Code

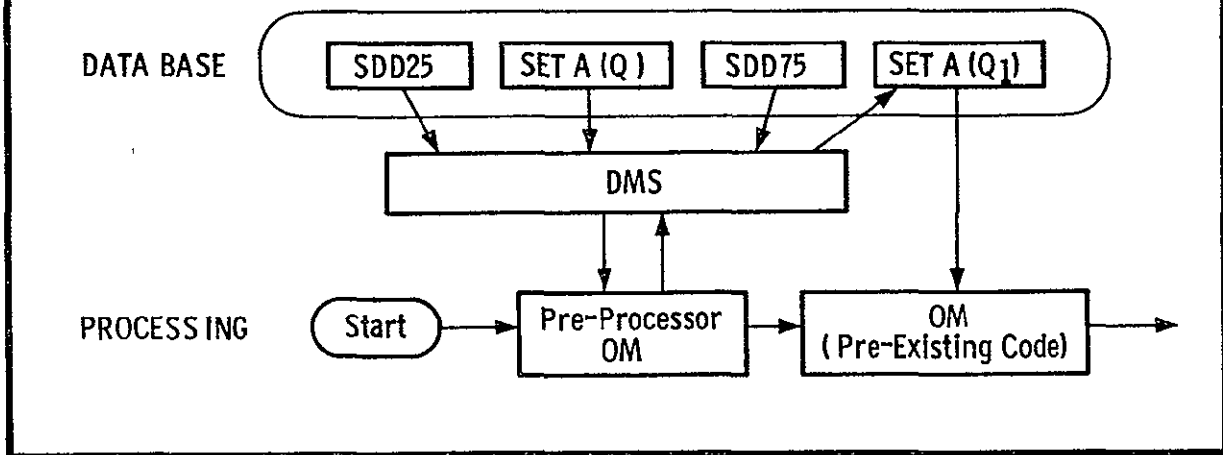


Figure 5.11 Example of Explicit and Implicit I/O Operations

Explicit I/O - When the data handling logic is explicitly present in the source code, (as it is with all source code generated without data manager type functions available) the data manager's function is limited to module boundaries; i.e., the data manager will prepare the data linkage prior to execution and dispose of the data after execution. Knowledge of the library entry internal structure by the data manager is not required during program execution.

5.7 PRIVACY, SECURITY, CONTROL, INTEGRITY

Many of the design features of IPAD, such as the library facility, in themselves provide privacy, security, control, and integrity. Other features are specifically designed to support these requirements. All features supporting these requirements are described in the following paragraphs.

Subtask and Community Libraries - The community library contains all of the application modules and data generally available to the

- AN SDD DESCRIBES A DATA STRUCTURE WHICH IS USER'S MODEL OF REAL WORLD
- AN SDD FILED IN THE DATA BASE IS A TEMPLATE USED BY DMS WHEN ACCESSING DATA

USER: Find Item X in Data Set SETA(Q)

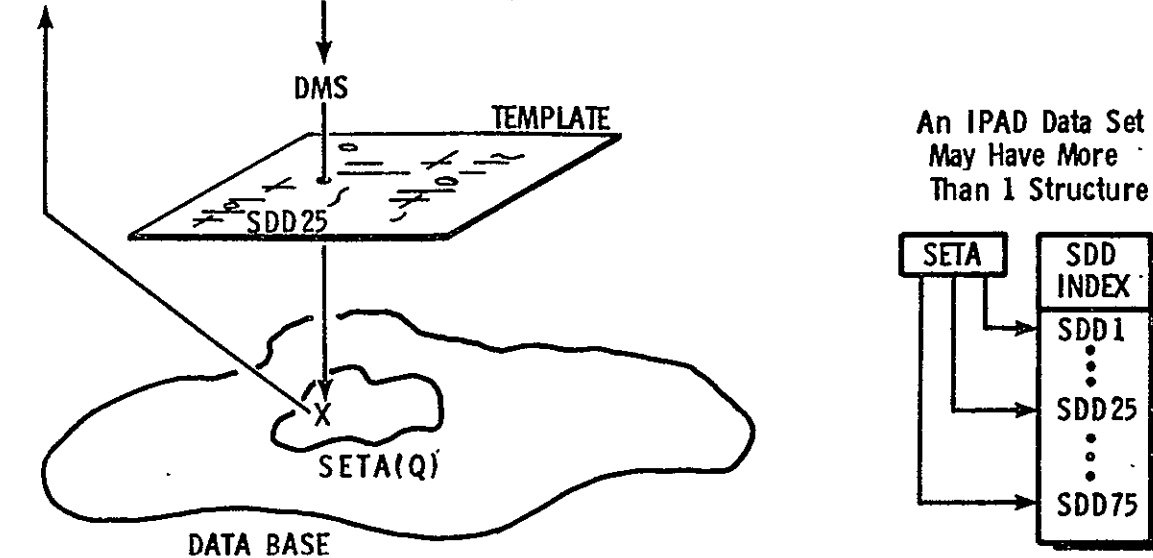


Figure 5.12 Stored Data Definition Illustration

community of users as a whole. The subtask library contains elements of the community library, and provides space for the user to do scratch work and otherwise prepare or generate data for entry into the community library. The subtask library may contain entries peculiar to an individual user and not available to the community at large. In this case, provisions are required to preserve the integrity or protect the quality of information being transferred to the community library.

Controlled Access to IPAD Data - Access will be controlled through access codes associated with individual entries of application code and data. For example all users could be allowed read permission for a data set but only particular users could be allowed write permission.

There will be at least five types of library entry access codes: read, write, extend, purge, and execute. Read access implies permission to read only. Write access implies permission to change existing information to a library text entry with corresponding changes in the library directory entry. Purge access implies permission to eliminate the entire library entry. Execute access implies that the entry may be executed but cannot be read for any other purpose.

The list of permission codes (i.e., identifiers for each user permitted each type of access) for each library entry will be listed in the access part of the directory. Each user's identification (UID) will be checked against these to determine the allowable access permission.

The above discussion relates only to the community library. Items originating in the subtask library are assumed to have all levels of access to the subtask owner. In the event the user requests write access to a community library entry for which he only has read permission, items may be copied to the subtask library in order to allow the task to continue without causing undesirable community library changes.

Controlled Access to IPAD System Commands - Use of IPAD system commands (see section 3.2) will be controlled through permission codes. A new user entering the system will be assigned a command profile internally within the system. This profile will define command and data regions the user may access. For example, except for exceptional circumstances, every user will be allowed to purge information from his subtask library but only specific users will be allowed to purge information from the community library. Specific permission will be required to send data outside IPAD. Transfer of information from a subtask library to the community library may be controlled to allow review before permitting the transfer. In tight security situations, purge permission from particular subtask libraries may be denied.

Uniqueness of Versions - Since IPAD is designed for large groups of users working on the same or related projects, it is necessary that they be able to change data sets and application modules without destroying the work or disrupting the plans of other users. Hence, any alteration of application modules or data within the community library will result in a new version. The root version will remain intact unless specifically purged by a user having permission. The assignment of new versions will be automatically required by the IPAD system whenever modifications are made.

Trace of Antecedents - A trace of the data sets and application modules used to generate a new data set will be compiled and preserved as qualifiers in the data set identification. This will allow users to trace the generation of a library entry.

Trace of Data or Code Leaving IPAD Control - A trace of data or code either purged from IPAD libraries or sent to a location outside IPAD control is required to protect the proprietary interests of the owner and to protect against sabotage, spying, maliciousness, or accidents.

Personal User Identification - Entry to IPAD will be via a personal user identification code to allow individual assignment of responsibility for certain acts and assignment of permission and access codes.

Reliability and Quality Controls - Where possible, the system should require conformity to standards and procedures that have been developed to ensure the reliability and quality of the system code, data and application modules. Further, a means of rating the reliability of new application modules should be provided according to the degree of checking that has been completed.

Protection Against Self-Inflicted Accidents - Protection against self-inflicted accidents will be made through the structure of the command language and by provision for recovery from command error where the action being taken has nonreversible consequences.

Security of the IPAD System Code - Where possible, provision will be made to control access to the IPAD system code itself to prevent tampering or unauthorized extensions.

Security of the Security Features Themselves - Careful consideration is necessary to restrict access to permission and access codes only to those persons authorized by the library owners.

Controlled Relationship Between Subtasks Within Projects - An important feature of the IPAD system will be the ability to input project planning as data by which lower level work can be monitored and regulated. Hence, a user responsible for a subtask may be informed of the relationship of his subtask to other subtasks forming the project.

Government or defense security provisions are provided by law or by requirement from the specific agency involved and are not considered here. Likewise, special security provisions necessary for a company to protect proprietary material are not considered.

6.0 TECHNICAL DESIGN SPECIFICATIONS

6.1 SYSTEM DESIGN METHODOLOGY

Structured Programming is a formal work dealing with software engineering and hardware-software system design and development (ref. 1, 2, and 3). The objective of this work is to transform the development of computer systems from a seat-of-the-pants art, to a disciplined technology. This approach has been utilized to develop the IPAD system design.

The structured programming approach is a top down design method in which the design proceeds from the general to the specific. Each refinement is a level in the system design. Tree structure diagrams give the system functional components in levels of increasing detail. The nodes at any one level in the tree structure are states of activity for the system. The entire system is included in the total set of nodes at each level, and in fact, higher level nodes are summaries of lower level nodes.

Transition diagrams describe how the system components, at each level, are functionally related. The diagrams also specify the conditions under which there will be a transition or state change within a node or from one node in the tree to another node at the same level. These transition conditions are (1) the input data or conditions that trigger the transition and (2) the output data or results existent in the system at the time the transition is made. Figure 6.1 is a sample tree structure and transition diagrams for a three level system.

The IPAD system design given in section 6.2 and Appendix A follows the general form described above. For level 1, twenty-nine nodes or states are described. Except for those level 1 states dealing with hardware or host operating system protocol, the level 1 states are each refined into level 2 states. The level 2 states are, in turn, broken out into level 3 states, and so on. The emphasis in the design was placed upon consistency in detail rather than consistency in levels documented. Hence, there are differences in the depth or number of levels reached in some of the tree branches.

While the design is presented in top down form, the actual design process does not proceed monotonically. Generally, design at level n will result in a review of some elements of the design at level $n-1$, $n-2$, etc. The advantage of the method is that the examination of effects is an orderly process and the consequences of the iterative design process are highly visible.

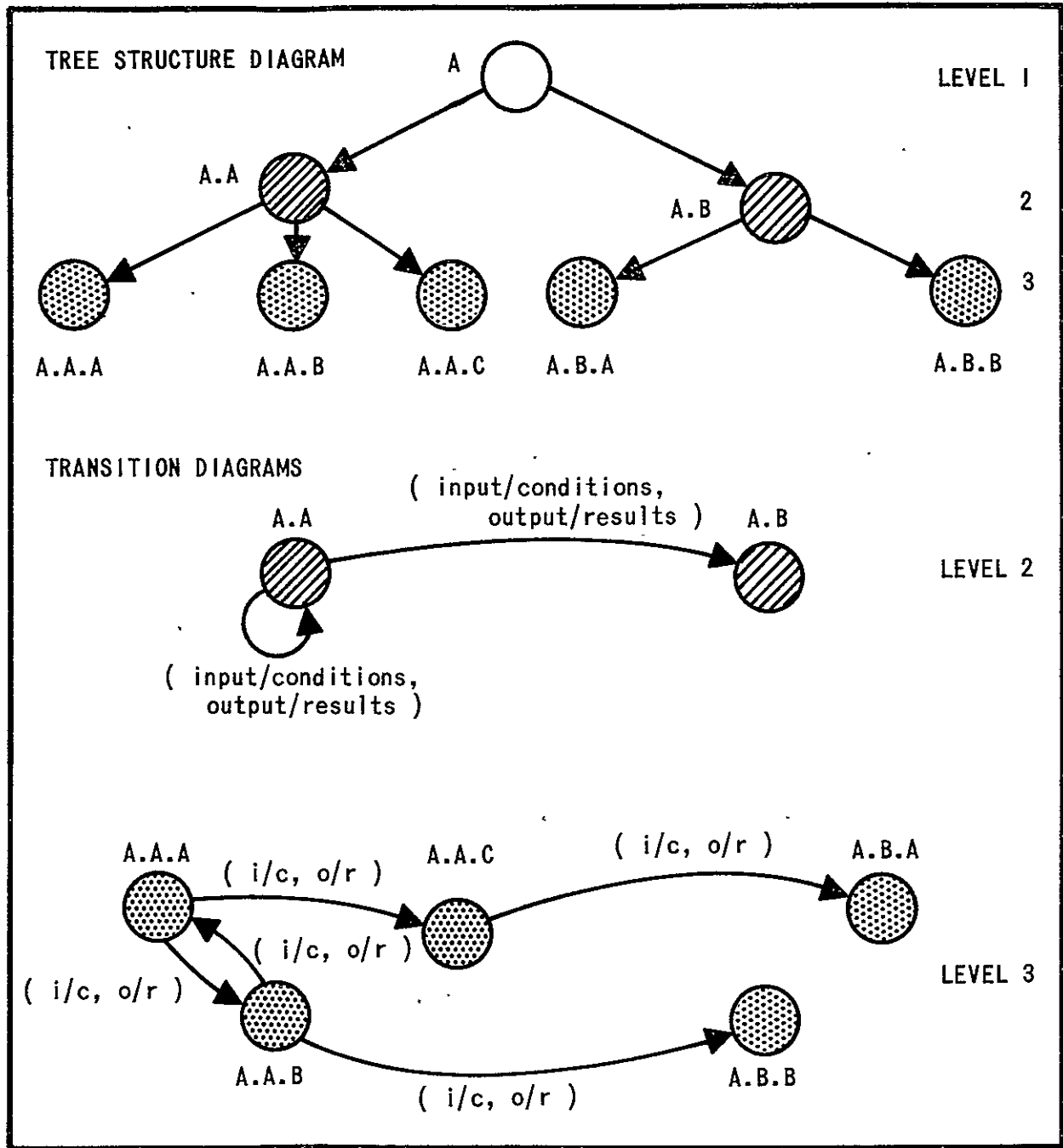


Figure 6.1 Structured Programming Diagrams

Level 1 nodes are included in section 6.2. Level 1 and all lower level nodes are included in Appendix A. Appendix A is divided by level; i.e., all information is given for level 1, then for level 2, and so forth. Level 1 in section 6.2 and each level in Appendix A contains the following diagrams and tables:

State Description Tables--Three pieces of information are given for each node.

- a) Short Structured Name--This name consists of a set of one or two alphabetic characters catenated in the form:

```
rs
rs. tu
rs. tu. rw
etc.
```

The syllable position denotes a level.. For example, if node A is at level 1 then node A.B would be at level 2 and would be a state of node A. Hence, the tree diagram can be formed from the short structured names. There is no requirement that these names be in sequence, i.e., the existence of node A.B and node A.D does not presuppose the existence of node A.C.

- b) Long Name--This name is descriptive of the function of the node. For example, node E has the long name "Subtask Set-Up."

- c) Description--Several sentences describing the capabilities of the node.

Allowed Transition Tables--This is a tabular representation of the connections between nodes that have a common parent at the next higher level. The states from which transitions are made, along with the corresponding references to the input/condition and output/result tables, which follow, are given. A bent arrow is used to flag entry and exit points from the parent node. When exits are shown, the level of the state exited to may be at a higher level than the state being exited from, depending upon the level of tree structuring completed.

Transition Diagrams--These are a graphical representation of the Allowed Transition Tables. They can be constructed from the transition tables and are valuable for visualizing relationships.

Input/Conditions List Tables--This is a list of the input or conditions that trigger a transition or change of state. This

list should be used in conjunction with the Allowed Transition tables.

Output/Result List Tables--This is a list of the output or results that are, existent in the state when a transition is made. This list should also be used in conjunction with the Allowed Transition tables.

Tree diagrams are not included. They can be constructed from the structured names.

Abbreviations are not used in level 1. They are used in lower levels to facilitate writing. Definitions of abbreviations are given in section 4.0. The text part of section 6.2 and Appendix A was created by a computer program from data supplied by the system designers. This computer program checked to ensure that transitions were made between valid states and that lower level states were correctly referenced to higher level states.

6.2 SYSTEM DESIGN SPECIFICATIONS

Figures 6.2 and 6.3 are the level 1 transition diagram. Node F is repeated on figure 6.3 for reference. These figures should be read in conjunction with the State Descriptions, Allowed Transitions, Input/Condition List and Output/Result List following the diagram. In some cases, the nodes at level 1 are more generalized functions than those given in Volume-III and in section 3.2 of this document. The association is given in table 6.1.

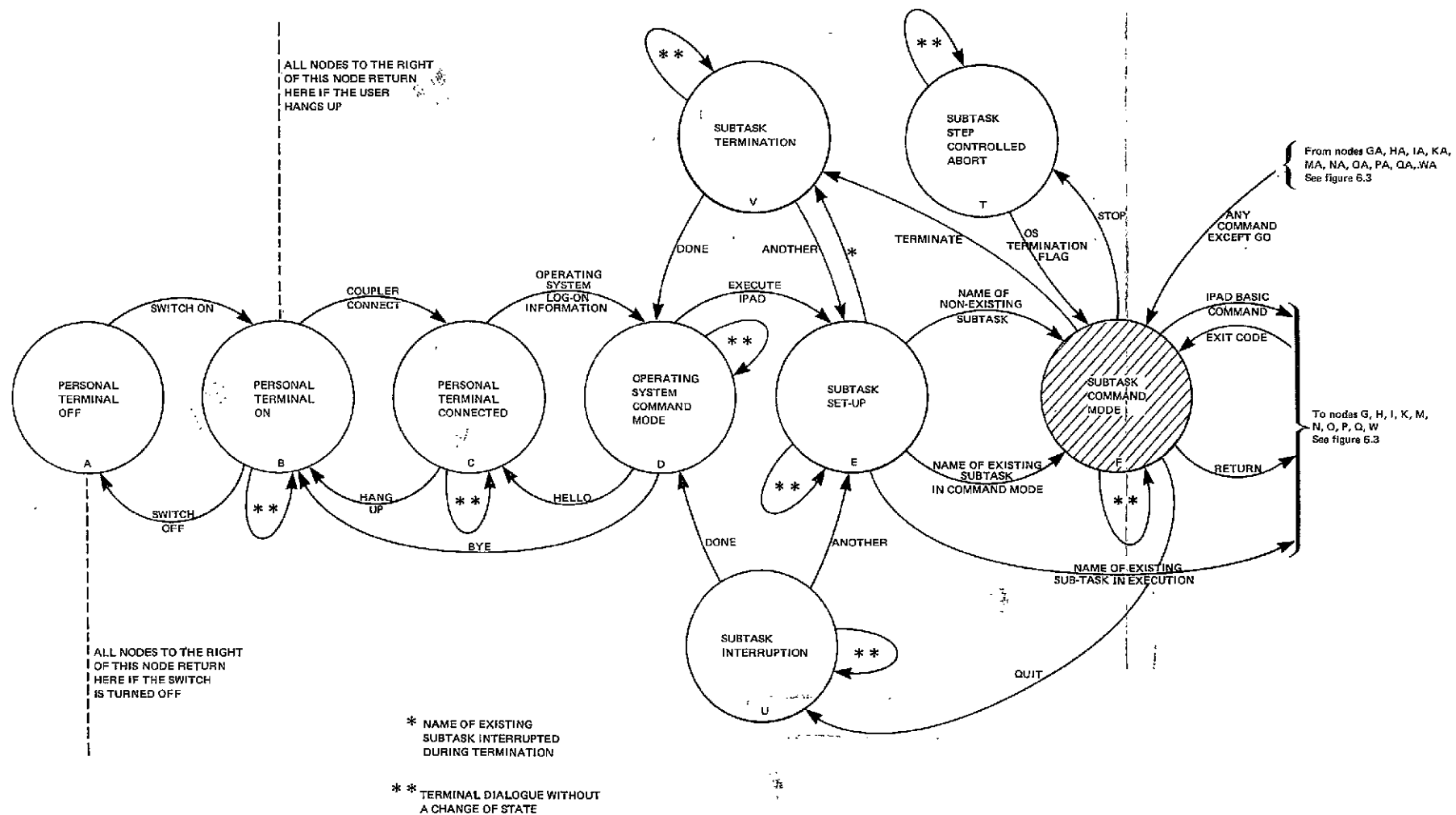
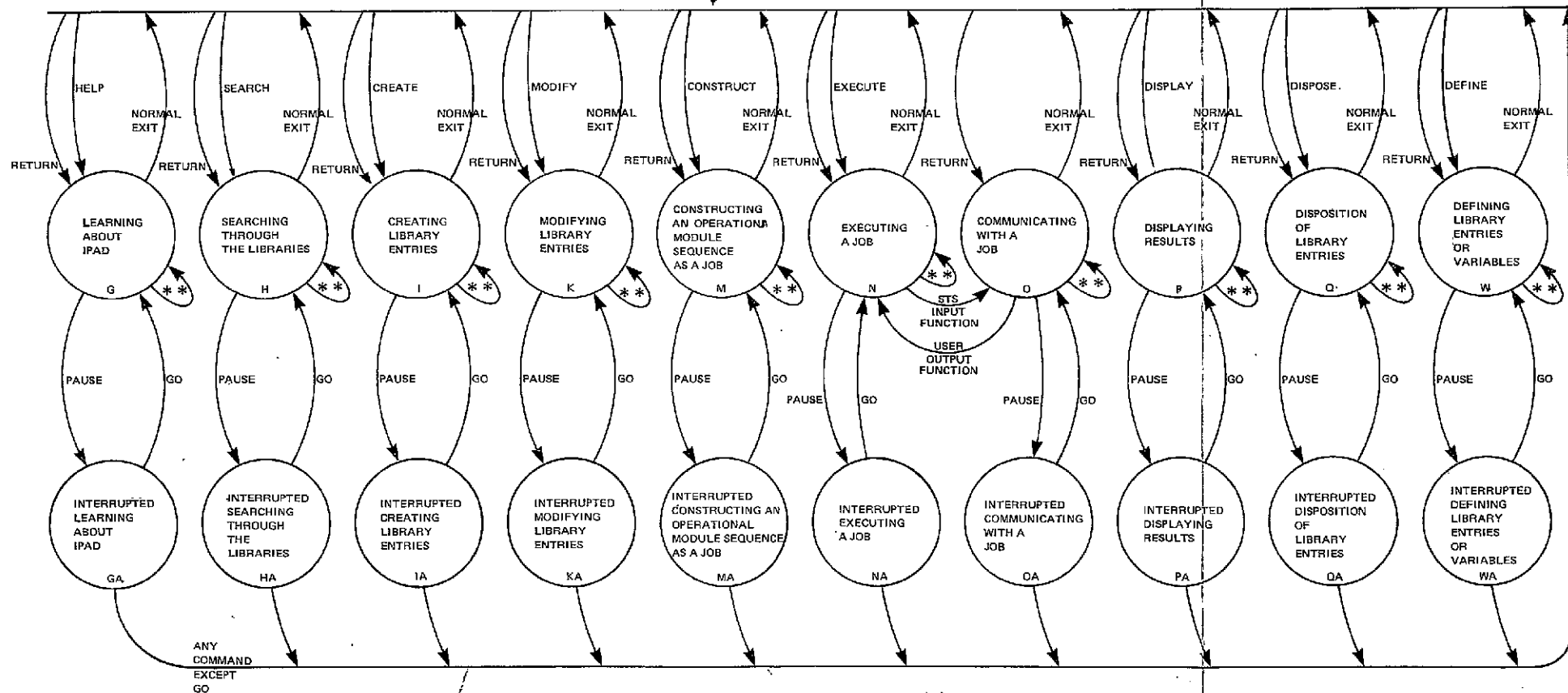
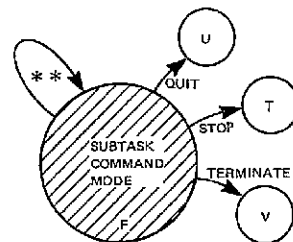


Figure 6.2 IPAD System Design Level Transition Diagram



** TERMINAL DIALOGUE WITHOUT
A CHANGE OF STATE

Figure 6.3 IPAD System Design Level 1
Transition Diagram. (Cont'd)

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

A PERSONAL TERMINAL OFF

THE EQUIPMENT IS NOT ACTIVE.

P PERSONAL TERMINAL ON

THE EQUIPMENT IS ACTIVE BUT NO DATA PATH TO THE COMPUTER EXISTS. THE EQUIPMENT IS A PERSONAL TERMINAL, NOT A REMOTE JCB ENTRY TERMINAL, BUT MAY BE AUGMENTED WITH PERIPHERAL DEVICES SUCH AS CASSETTE TAPE, PRINTER, PLOTTER.

C PERSONAL TERMINAL CONNECTED

THERE NOW EXISTS A TWO-WAY DATA PATH BETWEEN THE TERMINAL AND THE COMPUTER

D OPERATING SYSTEM COMMAND MODE

THE USER IS NOW ABLE TO ENTER COMMANDS TO THE TIME SHARING SYSTEM IN THE HOST OPERATING SYSTEM

E SUBTASK SET-UP

THE USER IS NOW IN COMMUNICATION WITH IPAD AND HE IS EITHER INITIATING A NEW SUBTASK OR CONTINUING AN OLD ONE. IN EITHER CASE, THE NET RESULT WILL BE THE ESTABLISHMENT OF HIS ACTIVE SUBTASK LIBRARY.

F SUBTASK COMMAND MODE

THE USER IS NOW ABLE TO ISSUE IPAD BASIC COMMANDS TO ADVANCE HIS SUBTASK WORK.

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE LONG NAME AND TEXT

G LEARNING ABOUT IPAD

THE ACTIVITY OF GAINING INFORMATION ABOUT IPAD
EITHER AS A TAUGHT COURSE OR AS HELP WITH A SINGLE
COMMAND OR MODULE.

H SEARCHING THROUGH THE LIBRARIES

THE PROCESS OF SCANNING DICTIONARIES AND DIRECT-
ORIES TO IDENTIFY AND LOCATE INFORMATION IN THE IPAD
DATA BASE.

I CREATING LIBRARY ENTRIES

THE PROCESS OF INSERTING DATA (NUMERICAL AND OTH-
ER) INTO THE IPAD DATA BASE RESULTING IN NEW LIBRARY
ENTRIES(LE). INCLUDED IS THE ENTERING OF SOURCE CODE
FOR CODING MODULES(CM), INFORMATION FOR STORED DATA DEF-
~~INITIONS(SDD), INSTANCES OF DATA SETS(DS), DISPLAY MENUS~~
(DI), AND THE INSTANCE OF THE SYSTEM DATA SET CONTAIN-
ING ACCESS AND PERMISSION CODES.

K MODIFYING LIBRARY ENTRIES

ALTERING CURRENTLY RESIDENT LIBRARY ENTRIES. THIS
CAN INVOLVE CHANGES TO ANY VALID IPAD LIBRARY ENTRY
TYPE.

M CONSTRUCTING A JOB

ARRANGING AVAILABLE CODING MODULES(CM) INTO OPER-
ATIONAL MODULES(OM), OPERATIONAL MODULES INTO JOBS, AND
OPERATIONAL MODULES AND PREVIOUSLY DEFINED JOBS INTO
NEW JOBS.

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
N	EXECUTING A JOB
.	ACTIVATING A PREVIOUSLY CONSTRUCTED JOB
O	COMMUNICATING WITH A JOB
	BEING INTERACTIVE WITH A USER CONSTRUCTED JOB
P	DISPLAYING RESULTS
	SCANNING, CHECKING, AND INTERROGATING INFORMATION CONTAINED IN LIBRARY ENTRIES OF ANY TYPE.
O	DISPOSITION OF LIBRARY ENTRIES
	TRANSFERRING LIBRARY ENTRIES BETWEEN IPAD LIB- RARIES, SENDING ITEMS OUTSIDE OF IPAD(OFFLINE, OR VIA A COMMUNICATION NETWORK), AND REMOVAL OF UNWANTED LIBRARY ENTRIES FROM THE DATA BASE.
T	SUBTASK STEP CONTROLLED ABORT
	THE TERMINATION OF THE CURRENTLY INTERRUPTED SUB- TASK STEP.
U	SUBTASK INTERRUPTION
	ACTION AIMED AT TEMPORARY INTERRUPTION OF THE SUB- TASK ACTIVITIES WITH THE INTENT OF RE-STARTING AT A LATER TIME AT THE PRECISE POINT OF INTERRUPTION.

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
-------	--------------------

V	SUBTASK TERMINATION
---	---------------------

THE USER HAS COMPLETED THE DEFINED SUBTASK AND NOW DESIRES TO DISPOSE OF ALL REMAINING INFORMATION, LOG THE TERMINATION IN THE PROJECT PLANS, AND ISSUE ANY REQUIRED REPORTS.

W	DEFINING LIBRARY ENTRIES OR VARIABLES
---	---------------------------------------

A DEFINITION IS A DICTIONARY ENTRY WHICH CONTAINS THE MEANING OF A VARIABLE OR A LIBRARY ENTRY AND CROSS REFERENCING INFORMATION. ALL COMMUNITY LIBRARY ENTRIES AND VARIABLES REFERENCED IN DATA SETS REQUIRE DEFINITIONS. DICTIONARY ENTRIES ARE OPTIONAL FOR SUBTASK LIBRARY ENTRIES.

GA	INTERRUPTED LEARNING ABOUT IPAD
----	---------------------------------

THIS IS THE STATE IMMEDIATELY FOLLOWING A PAUSE DURING LEARNING ABOUT IPAD. EACH OF THE STATES G, H, I, K, M, N, O, P, Q, AND W HAVE A SIMILARLY ASSOCIATED STATE.

HA	INTERRUPTED SEARCHING THROUGH LIBRARIES
----	---

IA	INTERRUPTED CREATING LIBRARY ENTRIES
----	--------------------------------------

KA	INTERRUPTED MODIFYING LIBRARY ENTRIES
----	---------------------------------------

MA	INTERRUPTED CONSTRUCTING A JOB
----	--------------------------------

NA	INTERRUPTED EXECUTING A JOB
----	-----------------------------

ORIGINAL PAGE IS
OF POOR QUALITY

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
DA	INTERRUPTED COMMUNICATING WITH A JOB
PA	INTERRUPTED DISPLAYING RESULTS
QA	INTERRUPTED DISPOSITION OF LIBRARY ENTR.
WA	INTERRUPTED DEFINING LIBRARY ENTRY/VAR

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
A	B	1	1
B	A	14	14
	C	2	2
C	A	14	13
	B	13	13
	D	3	3
D	A	14	13
	B	13	13
	C	15	15
	D	12	12
	E	4	4
E	A	14	20
	B	13	16
	F	5	5
	F	6	6
	V	16	6
F	A	14	17
	B	13	17
	G	17	22
	G	34	39
	H	15	22
	H	34	39
	I	13	22
	I	34	39
	K	21	22
	K	34	39
	M	23	22
	M	34	39
	N	24	22
	N	34	39
	O	34	39
	P	27	22
	P	34	39
	Q	26	22
	Q	34	39
	T	9	9
	U	8	8
	V	7	7
	W	28	22
	W	34	39

ORIGINAL PAGE IS
OF POOR QUALITY

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
G	A	14	40
	B	13	40
	F	35	42
	GA	31	36
H	A	14	40
	B	13	40
	F	35	42
	HA	31	36
I	A	14	40
	B	13	40
	F	35	42
	IA	31	36
K	A	14	40
	B	13	40
	F	35	42
	KA	31	36
M	A	14	40
	B	13	40
	F	35	42
	MA	31	36
N	A	14	40
	B	13	40
	F	35	42
	NA	31	36
O	A	14	40
	B	13	40
	F	35	42
	OA	31	36
P	A	14	40
	B	13	40
	F	35	42
	PA	31	36
Q	A	14	40
	B	13	40
	F	35	42
	QA	31	36
T	A	14	10
	B	13	10

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
U	A	14	19
	B	13	19
	D	11	11
	E	10	10
V	A	14	21
	B	13	21
	D	11	11
	E	10	10
W	A	14	40
	B	13	40
	F	35	42
	WA	31	36
GA	A	14	40
	B	13	41
	F	33	30
	G	32	37
HA	A	14	40
	B	13	41
	F	33	38
	H	32	37
IA	A	14	40
	B	13	41
	F	33	38
	I	32	37
KA	A	14	40
	B	13	41
	F	33	38
	K	32	37
MA	A	14	40
	B	13	41
	F	33	30
	M	32	37
NA	A	14	40
	B	13	41
	F	33	38
	N	32	37
OA	A	14	40
	B	13	41
	F	33	38
	O	32	37

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
PA	A	14	40
	B	13	41
	F	33	38
	P	32	37
QA	A	14	40
	B	13	41
	F	33	38
	Q	32	37
WA	A	14	40
	B	13	41
	F	33	38
	W	32	37

IPAD LEVEL ONE
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SWITCH TURNED ON
2	DIAL UP
3	VALID OS LOG ON INFORMATION IN THE PROPER SEQUENCE
4	VALID OS COMMAND TO EXECUTE IPAD
5	VALID SUBTASK IDENTIFIER FOR A NON EXISTING SUBTASK
6	VALID SUBTASK IDENTIFIER FOR AN EXISTING SUBTASK
7	TERMINATE
8	QUIT
9	STOP
10	ANOTHER
11	DONE
12	HELLO
13	USER HANGS UP
14	SWITCH TURNED OFF
15	BYE
16	SUBTASK RECORDS SHOWING AN INTERRUPT OCCURRED DURING THE SUB TASK TERMINATION
17	HELP
18	SEARCH
19	CREATE
20	DEFINE
21	MODIFY
22	CONSTRUCT
23	EXECUTE
24	CONDITION CODE SHOWING TERMINAL INPUT IS REQUIRED
25	LAST LINE OF USER INPUT
26	DISPLAY
27	DISPOSE
28	PAUSE
29	GO
30	ANY COMMAND EXCEPT A GO
31	RETURN
32	EXECUTION COMPLETED - NORMAL EXIT

ORIGINAL PAGE IS
OF POOR QUALITY

IPAD LEVEL ONE
(CONTINUED)

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	PHONE LINE CONNECT
3	VALID OPERATING SYSTEM LOG-ON INFORMATION
4	OPERATING SYSTEM COMMAND TO EXECUTE IPAD LOG-ON PROGRAM
5	ESTABLISHMENT OF A NEW SUBTASK LIBRARY IN THE CL
6	THE OLD SUBTASK LIBRARY IN ACTIVE FORM
7	OS COMMAND TO EXECUTE THE SUBTASK TERMINATION PROGRAM
8	OS COMMAND TO EXECUTE THE SUBTASK INTERRUPTION PROGRAM
9	OS COMMAND TO EXECUTE THE SUBTASK STEP INTERRUPTION PROGRAM
10	SUBTASK INTERRUPTION COMPLETE
11	-
12	VALID LOG OFF INFORMATION
13	PHONE LINE DISCONNECT
14	-
15	-
16	SUBTASK LIBRARY ENTRY RESTORED TO ORIGINAL STATE
17	A PROCEDURE WILL BE EXECUTED EQUIVALENT TO THE FOLLOWING INPUTS - 3,13.
18	COMPLETION OF TERMINATION, THEN PROCEEDING PER OUTPUT 17
19	COMPLETION OF INTERRUPTION, THEN PROCEEDING AS IF INPUT 13 HAD BEEN RECEIVED.
20	OUTPUTS 16 AND 13
21	HOLDING OF THE TERMINATION INTACT SO IT WILL BE ENTERED UPON RETURN, THEN PROCEEDING AS IF INPUT 13 HAD BEEN ENCOUNTERED
22	PARSED COMMAND, UPDATED ACTIVITY RECORD
30	EXPLANATORY TERMINAL OUTPUT(IF NEEDED), INPUT REQUEST
31	INPUT TO SUBTASK STEP
36	CURRENT SUBTASK STEP INTERRUPTED IN RE-STARTABLE FORM
37	-
38	POINTER TO INTERRUPTED SUBTASK STEP PLUS RESTART INFORMATION
39	OS COMMAND TO RE-START FROM PRECEEDING PAUSE
40	IPAD PROCEDURE EXECUTED CONSISTING OF A PAUSE, QUIT, DONE,BYE
41	IPAD PROCEDURE EXECUTED CONSISTING OF QUIT,DONE,BYE
42	NORMAL EXIT CODE

Table 6.1 IPAD System Design/Volume III Requirements
Comparison Summary

DESIGN NODE	DESIGN COMMAND	CORRESPONDING COMMAND FROM VOL. III
E	Name of Existing Subtask Name of Non-Existing Subtask	RESUME (from HOLD) No explicit command given for the initial log-on for a subtask
F	QUIT STOP TERMINATE RETURN HELP SEARCH CREATE MODIFY CONSTRUCT EXECUTE DISPLAY DISPOSE DEFINE	HOLD STOP No explicit command given for the ending of a subtask RESUME (from PAUSE) LEARN IPAD FIND ENTER EDIT CONSTRUCT, ENTER EXECUTE DISPLAY, FIND, COMPARE PURGE, SEND, TRANSFER DEFINE
G, H, I, K, M, N, O, P, Q, W }	PAUSE GO	PAUSE RESUME (after PAUSE)
None	None	MESSAGE

6.3 SYSTEM MEMBER SPECIFICATIONS

6.3.1 Design Organization

Section 6.2 contains the design specifications developed using a top down approach. Starting with basic user functions at the highest level, the various downward trails or "tree walks" specify the actions and data flow required by the system to carry out these functions. Viewing the complete system from another perspective, four basic operational elements or system members are identified whose relationships are shown in figure 6.4. These elements are:

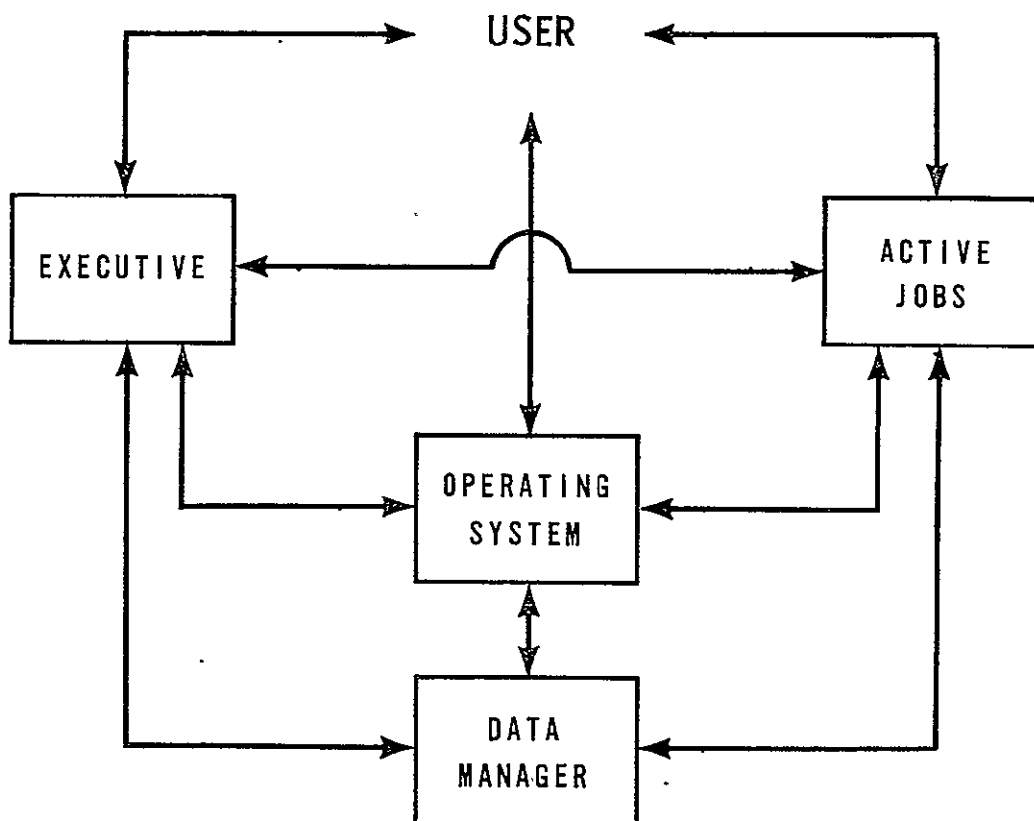


Figure 6.4 IPAD System Member Relationships

- a) ACTIVE JOB--A set of computer operations performing work for a user.
- b) OPERATING SYSTEM--The collection of software which controls the host computer and provides the interface for the non-IPAD user.
- c) IPAD DATA MANAGER--The collection of software unique to IPAD which controls and manages access to data in the IPAD data base.
- d) IPAD EXECUTIVE--The collection of software unique to IPAD which interprets user commands and controls and manages system activities in response to those commands.

No attempt has been made to map the design nodes of section 6.2 and Appendix A completely into the system members. Table 6.2 illustrates how this mapping might be done. Identifying nodes with system members is primarily an implementation task.

The concept of community and subtask libraries and the underlying system data structures is basic to the IPAD design. Implementation guidelines for the IPAD Executive and Data Manager are given in sections 6.3.2 and 6.3.3. Specifications of the data structures are given in section 6.3.4.

6.3.2 IPAD Executive

A design goal has been to minimize the IPAD interface with any given host hardware/software system. With regard to the IPAD executive this implies that it should execute as an ordinary job under the host operating system. The degree to which this is not true is a measure of the host system dependency of any given IPAD executive implementation.

The most significant result of these considerations is the design feature of one executive program per user. If the IPAD executive is to look like a user program and there is to be one per user, at least three demands are put on the host operating system:

- It must contain a time sharing system with normal terminal input/output handling features.
- It must have a multitasking capability with the ability for one task to interrupt another.

Table 6.2 IPAD System Member Mapping

SYSTEM MEMBER	DESIGN NODE(S)	EXPLANATION
EXECUTIVE	E F T U V	Subtask setup Subtask command mode Subtask step controlled abort Subtask interruption Subtask termination
ACTIVE JOBS	G,H,I,K,M, N,O,P,Q,W	All utility operations and user application modules.
DATA MANAGER	K.C.B.	Update usage information This is an example to show that the data manager enters the system design at levels 2,3, and below. The data manager is support to the executive and active jobs and therefore will not be visible to level 1. Most nodes at a lower level will use the data manager in same way.
OPERATING SYSTEM	N.D.A	Initiate execution The operating system supports IPAD; therefore, the interface to it appears at the lower design levels.

- It must be able to retain the linkage between a user and his executing job without maintaining a terminal connection.

The first of these requirements must be satisfied to support personal terminals in IPAD. The second requirement must be satisfied if the executive is to initiate IPAD utilities and user jobs, interrupt them, and restart them. The PAUSE command (see figure 6.3) may be issued by the user at any time during execution requiring the operating system to suspend execution of the current activity and initiate a new activity. Repeated interruptions are allowed, giving rise to multiple activities in suspension associated with a single user/terminal combination. Since an interrupted activity must be restartable, operating system functions like roll-out and roll-in will be callable by the IPAD executive. The third requirement is necessary to free the user and his terminal during long executions. If an execution requires more than a few minutes, the user may desire to pursue other tasks and inquire at a later time about the progress of his executing job.

6.3.3 Data Manager

6.3.3.1 Design Intent

The IPAD data manager will consist of a set of computer programs written in IPADL (see section 6.7) and underlying software provided by the host system. The IPADL programs are intended to be machine independent but will be dependent on a level of data base management software similar to that specified by the CODASYL Data Base Task Group (DTBG) in their report of April 1971, (ref. 4). If the host system does not include software supporting a CODASYL defined data management system it is recommended that it be added and used. Other software providing the same capability would suffice. At the IPADL level of implementation the underlying software should be invisible.

A fundamental concept in the IPAD data management system is the separation of definitions of data structures from programs that reference the data. The IPAD stored data definitions are the basis for all IPAD controlled data functions as well as the means by which implicit I/O is provided to user programs. The schema and subschema of CODASYL are stored data definitions, and in the following sections modifications to the CODASYL specifications necessary for IPAD are given.

6.3.3.2 Modifications to CODASYL Data Description Language (DDL) Specifications

External Data Structures--Provision for the definition of data structures which reside on storage media such as punched cards and magnetic tape are necessary for some IPAD utility functions dealing with the movement of data into and out of IPAD. Subschema should permit the differentiation of external and internal structures and include facilities for storage device and media control information. Examples are: field definitions on punched cards, tape blocking factors and tape mode and density. The DDL, as specified in the April 1971 CODASYL report of the DBTG, does not handle these problems but perhaps will by the time IPAD implementation is begun. Additional work currently under way in CODASYL by a new group, the Stored Data Definition and Translation (SDDT) Task Group, is dealing with this problem. Papers were presented by this group at the SIGFIDET Conference on Data Description held in Denver, Colorado in November 1972, (ref. 5). The development produced by the SDDT Task Group should be evaluated as an early part of the IPAD implementation effort.

Local Variable - Library Variable References--The CODASYL DDL specifications must be modified to allow subroutines to reference a global variable with their own local variable names. CODASYL DDL statements for a schema include those for defining variables as data subentries of record entries. CODASYL also permits the definition of subschema which specify which parts of a data set the program declaring the subschema may access. In the DDL for a subschema there is a RENAMING section in which variables and aggregates of variables (e.g., records) can be given names local to a subschema. The CODASYL specifications, however, require that all routines which use a given subschema use the same local names. The RENAMING section must be augmented for IPAD to accommodate the coding module to operational module building block logic. The addition of a FOR phrase in the DATA renaming clause provides this augment for variables.

As an illustration:

DATA data-base-identifier-1 IS CHANGED TO data-base-data-name-1 FOR subroutine identifier-1 (, TO data-base-data-base-name-2 FOR subroutine identifier-2).....

FOR phrases can be similarly used to rename sets and records.

When a coding module is generated, the subschema required for implicit I/O are identified by each subroutine in the CM. Two or more subroutines declaring the same subschema may have different names for the same variable in the data set identified in the DATA statement by data-base-identifier-1. There is a

complimentary requirement (discussed in more detail below) that all local-global name references be correctly resolved at execution time. The modifications specified here for subschema renaming can be used directly for this resolution.

It should be noted that the default mode uses the same local and global variable names, and this mode will frequently be used when new programs are developed under IPAD. The renaming capability is needed, however, for incorporation of pre-existing and independently developed code into IPAD.

6.3.3.3 Data Manipulation Language (DML) for IPADL and the IPADL Compiler

As of this writing the CODASYL DBTG has only specified a COBOL DML. If CODASYL produces a FORTRAN DML, as planned, it should be relatively simple to adapt it to IPADL. If a FORTRAN DML is not available a DML for IPADL must be specified.

Assuming the resulting DML is modeled after the one which exists for COBOL, a FIND command issued by a program will cause the data manager to move the desired data from the data base storage device to a system buffer. A subsequent GET command moves all or part of the data from the system buffer into the user's working area where it is manipulated by the calling program. As mentioned above, different IPAD subroutines may reference a single global variable with their own local names. When an Operational module is executing and a GET command from a subroutine has moved some data into the user's working area (UWA); that portion of the UWA resembles a FORTRAN common block with respect to the data aggregate moved and the other subroutines which use the same subschema. Three conditions must be satisfied to provide proper referencing to the data in the UWA:

- a) The FIND (or equivalent command) must identify the requestor so the correct data aggregate is retrieved from the data base;
- b) The GET (or equivalent command) must identify the requestor so the correct data aggregate is moved to the UWA, and
- c) The correct address references are made to variables in the UWA from each subroutine using the same subschema.

These requirements suggest modifications to the DML such as adding a FOR phrase to the FIND, GET commands. This, however,

would only satisfy condition (a) and partly satisfy condition (b). Alternatively, the IPADL compiler could be designed to use the subschema at code generation time which enables it to use global names for all retrieval/storage commands, and to allocate space for data in the UWA. A mechanism, like FORTRAN labeled COMMON, would result from the compiler having obtained the block structure and the local name equivalences from the subschema. All local references, in effect, would then be transformed by the compiler into global references without any action by the subroutine other than to identify the proper subschema at compile time. The DML should, therefore, include subschema declarations.

6.3.4 Data Structures

The IPAD system uses a set of data structures organized into libraries as discussed in section 5.4. By convention, all data stored in the IPAD data base resides in library entries. Each library entry is divided into two parts, the library directory entry and the library text entry. The directory contains identification and control information pertaining to the text. The text is the information set stored. The directory/text relationship is essentially the same as an envelope/letter relationship. Figure 6.5 shows the basic elements of a library entry.

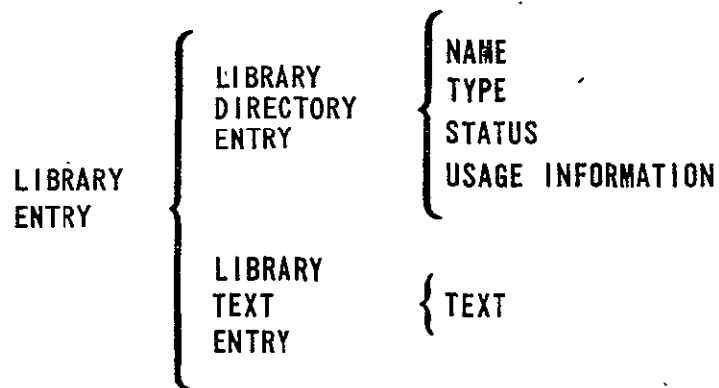


Figure 6.5 IPAD Prototype Library Entry

The structure of all library entries are defined in the system by stored data definitions (SDD) which are themselves library entries. By virtue of usage there are two classes of

PP = project pointer--linkage with the project (could be project or subtask).

b) TYPE--Must be one of the following codes

AL = Alias
CM = Coding Module
DY = Dummy
DF = Display Format
DC = Dictionary
DR = Directory
DM = Display Menu
DS = Data Set
OM = Operational Module
PL = Plan
RP = Report
SDD = Stored Data Definition
SJ = Standard Job (synonym for Job)
ST = Subtask
SF = Security File (see section 6.5 below).

c) STATUS--Basic Description One of the items under each of the following headings must be selected.

Availability - (available, purged, archive)

Security - (unclassified, confidential, secret, etc.)

Certification Level - (checkout, ..., certified)

Analysis Level - (1, 2, ...n)

Current Structure - (for any LE which may have text in optional formats, the name of the current SDD)

Current State of Access

For each user currently attached to the LE, the user's identification (UID) will be kept along with the type of access he is permitted to have.

Responsible Person/Organizatbon

Identification of "owner" of the LE.

External Document Reference

When applicable, references which are pertinent to the text associated with the LDE.

Text Control Data

Existence and contents of this category of information is type dependent. Specifications are given with the LTE specifications in section 6.3.6.

Installation Table

Reserved for host system dependent data which may be necessary for successful operation.

d) USAGE INFORMATION

Date, time or originating action with owner's identification.

Date, time and type of last access with UID

User access table

UID	READ	WRITE	EXECUTE	EXTEND
UID ₁	PW ₁ AC			PW ₁ AC
UID ₂	PW ₂ AC	PW ₂ AC		
UID ₃			PW ₂ AC	
ANY			PW ₄ AC	
ANY			ANY AC	

AC = Date of last access, total access count.

Note: For the ANY entries, there is the option to keep additional statistics by UID.

6.3.6 Library Entry Specifications

The contents and use of the LTE, by type, are specified below. The directory entries for some types of data sets contain a collection of control information specific to the type. This information, the Text Control Data, is also specified below. Table 6.3 contains a summary of the system structures.

a) ALIAS (AL)--The ALIAS type is provided as a convenience in resolving data set naming conflicts at the operational module and job level. An ALIAS may have a qualified or unqualified name, with or without versions. The ALIAS has no text of its own but points to another data set. This "parent" data set may be of any type and has appropriate text. All access permission is based on the aliased (parent) data set.

b) Coding Module (CM)--A CM is the basic building block for the production of executable programs in IPAD. CM names are unqualified library entry names with versions and the text contains source and object code. The formats of the source and

Table 6.3 IPAD System Structures

LIBRARY ENTRY USAGE DEFINITIONS				LIBRARY ENTRY = LE										LIBRARY TEXT ENTRY = LTE	
TYPE	DEFINITION and/or USE	LIBRARY ENTRY	NAME	LIBRARY DIRECTORY ENTRY = LDE							TEXT CONTROL DATA	TEXT			
				TEXT LOCATION	RESPONSIBLE PERSON	DOCUMENT REFERENCE	INSTALLATION	USAGE TABLE	TYPE CODE	STATUS					
ALIAS (AL)	Allow name substitutions to resolve ambiguities.	—	All Possible	Link to Parent	—	—	—	—	—	—	—	—			
CODING MODULE (CM)	Smallest package of computer code in system. Used as a building block to create executable programs. A CM may have more than one subroutine	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	Sub-routine Block for each S-R (See Section 6.3.6B)	(1) Source code (2) Object code			
DUMMY (DY)	Allow LE names to be used as formal parameters.	—	ULEN without versions	—	—	—	—	—	✓	—	—	—			
DISPLAY FORMAT (DF)	Allow identification of specific displays. May be a user supplied display processor or a parameterization procedure for a system supplied display function	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	(1a) OM or Job ID (user supplied), or (1b) System utility ID (2) List of LE names or types which may be displayed	Detailed format specifications or other control information required by display processor			
DICTIONARY (DC)	All definitions to be stored which give meaning to LEs in data base. Used for certain types of retrievals and user communication	✓	QLEN	Link to Text	✓	✓	✓	✓	✓	✓	—	Collection of dictionary entries (See Section 6.3.6E)			
DIRECTORY (DR)	Used to locate, manage, and control data (LEs) in data base. Consists of index in text + the LDEs linked to by the index	✓	QLEN	Link to Text	✓	✓	✓	✓	✓	✓	—	Index to a set of LDEs. Note that an entire library (CL or STL) or a part of a library may be referenced by a directory.			
DISPLAY MENU (DM)	Allow users to specify display options for a series of sessions	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	Display option table.			
DATA SET (DS)	Repository for data required by and produced by users. Used to coordinate multiple users working on a project	—	QLEN	Link to Text	✓	✓	✓	✓	✓	✓	—	Data values			
OPERATIONAL MODULE (OM)	Smallest package of executable code. Used as a building block when defining an execution sequence.	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	(1) List of Coding Modules. (2) Operational module specifications (3) Generated main program.			
PLAN (PL)	Used for project, task, subtask management.	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	(1) "Pert" chart with level codes (2) User control codes (3) Completion Action (4) Message buffer			
REPORT (RP)	Used for reports produced at subtask termination time	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	Report text.			
STORED DATA DEFINITION (SDD)	User supplied definition of a data set which contains sufficient information to allow data access by user programs and system utilities to be managed and controlled by DMS	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	Schema and sub-schema for the data set defined by SDD. (Schema use in CODASYL sense).			
STANDARD JOB (SJ)	The unit of execution in IPAD. User defines jobs as combination of OMs and/or other jobs.	✓	ULEN with versions	Link to Text	✓	✓	✓	✓	✓	✓	—	(1) Network description of job (2) "Symbol table". (3) Control card skeleton			
SUBTASK (ST)	Used by the system to manage a subtask activities and subtask library.	—	ULEN	Link to Text	✓	✓	✓	✓	✓	✓	—	(1) Activity Record (2) Data Set Reference Table (3) Termination Record			
SECURITY FILE (SF)	Used by the system to control access to Data base.	✓	ULEN	Link to Text	✓	✓	✓	✓	✓	✓	—	User security profiles			

ORIGINAL PAGE IS
OF POOR QUALITY

object code will be compatible with the host software provided for maintenance of character string data and for the loading and execution of compiled routines. A CM may contain more than one subroutine.

Text Control Data for CM--A block for each subroutine in the CM as follows:

- 1) Subroutine Name
- 2) Main Program Flag
- 3) Entry Point List
Name, OM Call Flag
- 4) External Reference List
CM Name, Entry Point Name
- 5) Common Regions Referenced and Dimensions
- 6) Data Control Specifications, for each data set referenced
 - Name: (ULEN)
 - Use: IN, OUT, I/O, Scratch
 - Mode: Implicit, Explicit
 - Set up: If mode - Implicit, subschema name
If mode - Explicit, file name, position
of data set on file, and file
- unit correspondence.

The information in this block cannot be completely specified until the programming language and host software are known. The specifications given assume a FORTRAN-like language.

c) Dummy (DM)--The dummy data set type is provided to allow data set names to be used as formal parameters at OM and Job definition time. Substitution of actual data set names for the dummy names takes place at execute time. There is no text associated with a dummy, and the name is a simple ULEN.

d) Display Format (DF)--This type of data set is used to identify specific display capabilities provided by the users such that the displays may be readily invoked. The display software may be user supplied for a system utility driven by a

procedure parameterized by the user. DF names are unqualified but may have version numbers. The LTE contains control information, format specifications, procedures (which may be a combination of IPAD commands and host OS control language statements).

Text Control Data for DF

1) Processor identification

- (a) User supplied Job or OM ID, or
- (b) System utility ID, or
- (c) Procedure flag .

2) List of LE names or types which may be displayed.

e) Dictionary (DC)--A dictionary name may be qualified. The fundamental purpose of a dictionary is to provide unique, unambiguous definitions of data items. The dictionaries are used to search the libraries, both through keywords and direct references. The LTE of a dictionary contains the individual sub-entries. A prototype follows:

Name (ULEN or Variable name)

Type

Defining Text

Documentation References

Responsible Person/Organization

Key Word List

"Used by" List

Variables used by Data Sets

Data Sets used by CMs

CMs used by OMs

OMs used by Jobs

f) Directory (DR)--A directory name may be qualified. The text of a directory contains an index which points to a set of LEs. A directory is used by the system to access the CL, and each STL has a directory. The users do not have direct access

to the CL and STL directories that are maintained and manipulated by the system.

g) Display Menu (DM)--Names of DMs are unqualified but may have versions. The LTE contains a display option table that is used to simplify requests for displays which the user may make frequently over the course of working a subtask.

h) Data Sets (DS)--Data sets contain user data in the LTE whose structures are defined by stored data definitions. Data set names are qualified and may have version numbers.

i) Operational Module (OM)--An OM is the smallest executable unit in IPAD and consists of one or more CMs. OM names are unqualified but may have version numbers. The LTE contains a list of the CMs. An OM must contain one and only one "main" program which is either included in one of the CM constituents or produced by the system at the time the OM LE is entered by the user. In this case the main program specifications are given in a high level IPAD language and the source statements are stored in a separate, newly defined CM.

j) Plan (PL)--The Plan is used to contain control information for a project. Plan names are unqualified but may have version numbers. The LTE has four main categories of data:

- 1) Description of subtasks and PERT type network,
- 2) A set of user control codes for each subtask. This information is used in conjunction with data from the system security file and permission codes in the LEs to control activities and data access at all levels.
- 3) Specifications of actions to be taken on completion for each subtask. These include references to Report LEs to be produced, the establishing of other subtasks, issuing messages to subtasks and to project management.
- 4) Message buffer used to pass coordinating information among active users on a day to day basis.

k) Report (RP)--Reports may have qualified names. The report is linked to another LE of type SJ (Standard Job) which contains the information required to produce a report at subtask termination time. This includes a definition of the contents, format, and data sources. The LTE of the Report LE contains the actual report produced by the referenced job.

l) Stored Data Definition (SDD)--SDD names are unqualified but may have version numbers. The LTE of the SDD contains the detailed specifications of the LTE being defined; i.e., the schema and subschema referred to in the CODASYL DBTG report.

m) Security File (SF)--The security information is contained in the LTE. The SF is a unique system LE in the community library. Each subtask library will also have a security file representing the total security profile for that subtask. Specifications of the LTE contents are contained in section 6.5.

n) Standard Job (SJ)--Job names are unqualified but may have versions. The job is the unit of execution in IPAD and is a combination of OMs and/or other jobs. Three categories of information are in the LTE.

- 1) Network description of job consisting of source statements of the job definition language.
- 2) A "symbol table" identifying logical file names used and the unqualified names of data sets which are external to the job.
- 3) An execution procedure which is a combination of OS. control cards and IPAD commands to be parameterized at run time.

o) Subtask (ST)--Subtask names are unqualified. There is one ST type LE in each subtask used to record activities and status of the subtask. The ST LE will only appear in subtask libraries, never in the community library. Three categories of information are contained in the LTE.

- 1) Activity Record--A complete record of all activities in the subtask such as IPAD commands processed and status. Accounting information showing resources (cost) used by activity.
- 2) LE Reference Table--This information is used to determine whether changes have been made to LE in the community library which are referenced by the subtask from session to session. Current line numbers of the usage information table (in the directory entry) of each LE are recorded at the beginning and end of each session. Gaps in the numbers between sessions indicate changes. Analysis of the usage data will help determine the effect on a particular subtask.

- 3) Termination Record--The specifications of the activities to be performed when the subtask is complete are contained in the LTE of the Plan. Each activity is logged here when it is started and the status recorded, through completion. This record becomes part of the project report.

6.3.7 Logical Organization of IPAD Libraries in Data Base

IPAD data management is based on the use of the system LEs to contain the different types of information held in the system. The total collection of data in the data base is divided into one public aggregate, the community library, and many private aggregates, the subtask libraries. With the exception of subtask and security file types, LEs of all types may be found in any library.

Access to the data base by the IPAD system is by way of a location in the host operating system which points to the directory of the community library. This directory is an index to all LEs in the CL. Each subtask has a directory in the CL, which is an LE of type "Directory" having the name of the subtask. The text entry of the subtask contains an index which points to all the LEs which comprise the subtask library. An LE in the community library may logically be attached to one or more subtasks, such attachments being shown in the directory entry of the LE. The index of each attaching subtask will in turn reference the CL entry.

Figures 6.6 and 6.7 offer two views of the library organization. The first illustrates the chaining of pointers which connects the total data base. The second shows what resides in the community library and subtask libraries. Note that the community library directory is the text entry for a library entry called DIRECTORY (CL). Also note that the subtask libraries consist only of text entries and that all directory information resides in the community library.

The effect of this organization is that subtask libraries are only partially visible in the CL. A scan of all the CL entries will not disclose any subtask library entries except those CL entries which are attached to subtasks, and the directories of the subtask libraries. Subtask libraries are thus seen to be referenced indirectly, one level down from the CL.

A relationship unique to data set type library entries is shown in figure 6.8. A data set library entry is composed of one or more library variables each of which must be defined in the library variable dictionary (see 5.4.2 and 5.4.3). This

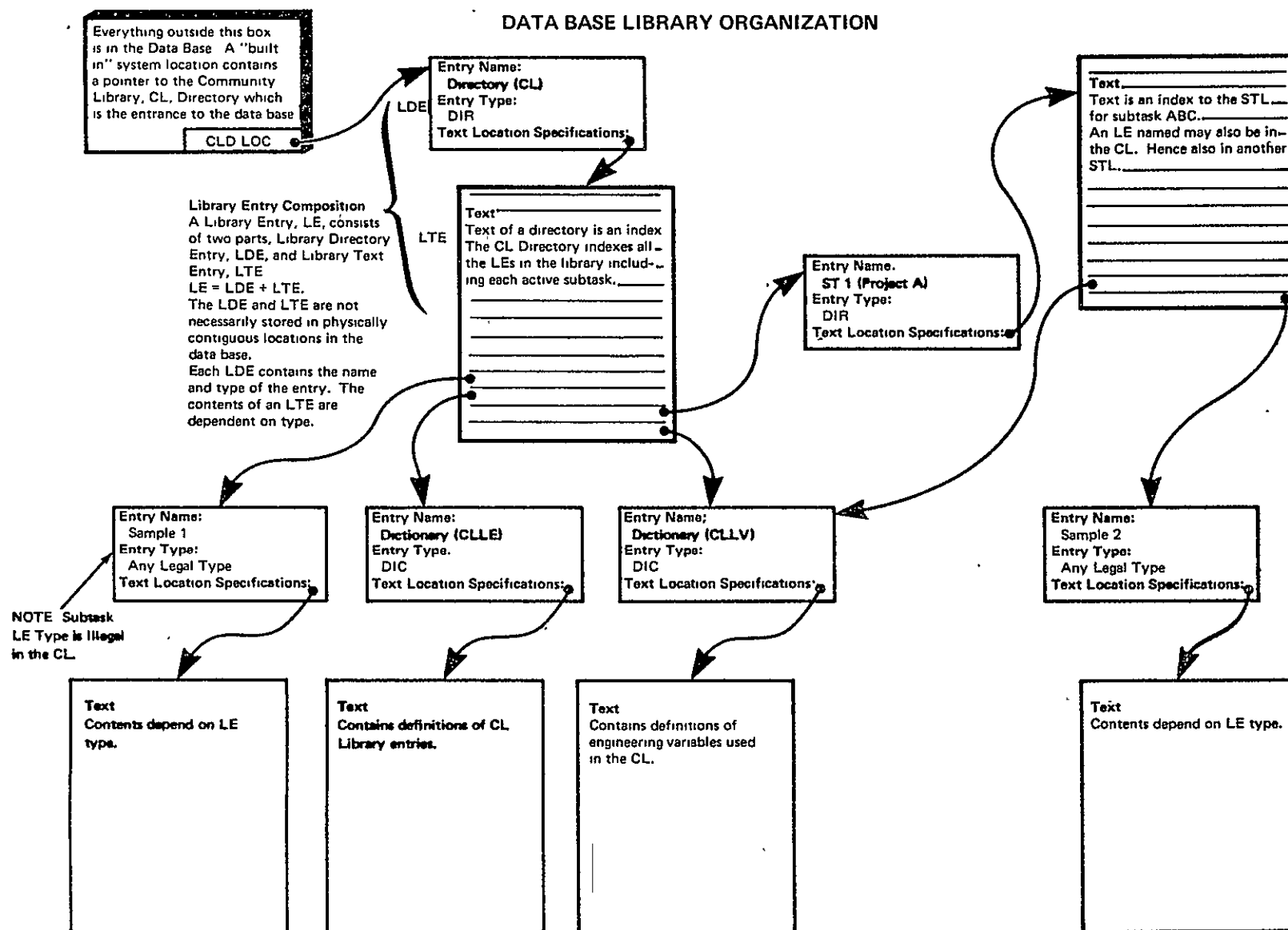
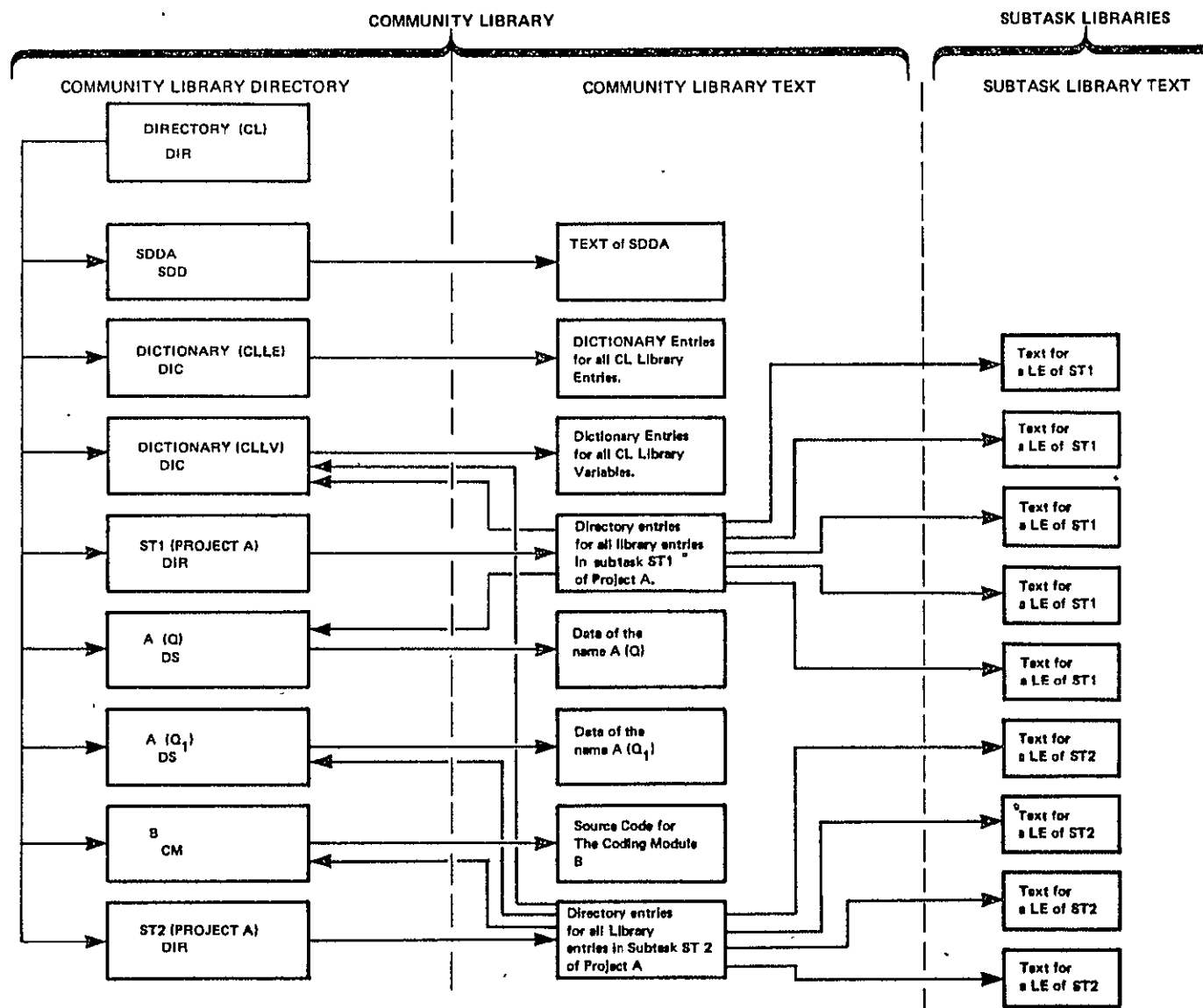


Figure 6.6 IPAD Data Base Library Organization

0-2



ORIGINAL PAGE IS
OF POOR QUALITY

Figure 6.7 IPAD Library Organization

collection of library variables is then defined in the library entry dictionary as a library entry. This then defines a conceptual data set which has an unqualified name but no actual data associated with it. A collection of data representing a particular instance of the defined data set has a qualified name in the directory. The directory has the linkage to the actual data.

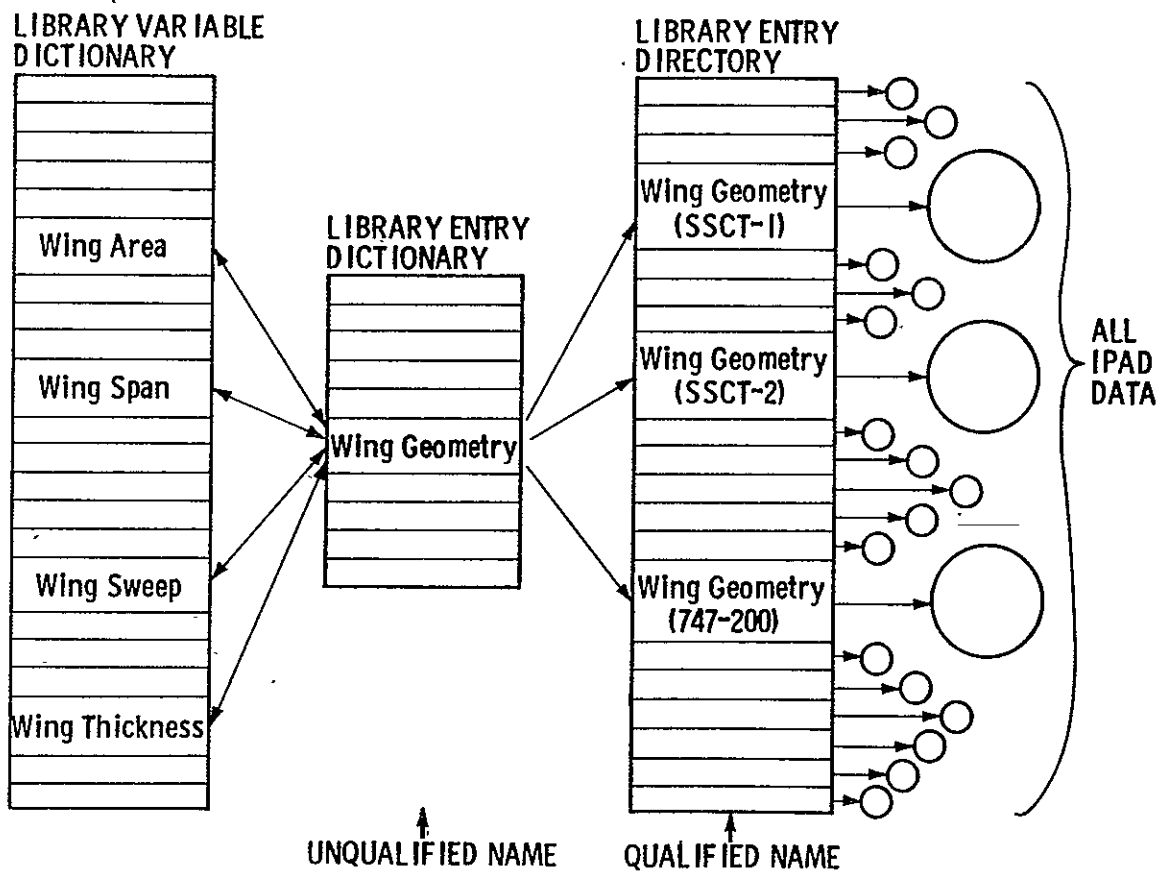


Figure 6.8 Data Set Library Entry Organization

6.4 HUMAN FACTORS

The characteristics of the man as well as the computer must be included in the design of a man-computer dialogue. The ability of man to adapt to a wide range of circumstances directs the designer of a man-computer dialogue to give greatest consideration to the least adaptive of the two--the computer. Too often the needs of the man are determined from a value-based definition which leads to the ultimate conclusion that the real needs of man are only associated with food, water, and shelter. A more useful basis is a rational definition wherein a "need or requirement is some demonstrably better alternative in a set of competing known alternatives that enable a human purpose or action to be implemented" (ref. 6). This definition deliberately ignores the argument of value versus cost, an argument that is never conclusive in the design of a man-computer dialogue. It does allow for an exploration of alternatives and their implications on the quality of work, efficiency, and general creativeness of man.

Language is the principal vehicle in a dialogue. Since man is the dominant element in the dialogue, the following three observations about the behaviour of man are pertinent:

- Behaviour is strongly time associated.
- Behaviour is conditioned by familiarity and expectation.
- Familiarity and expectation are the result of experiences.

These observations are developed as the basis for man-computer dialogue design in the following paragraphs.

6.4.1 User Behavioural Characteristics

Reduction in computer response time from several days to several minutes by going from a batch system to a terminal system may be an adequate improvement if the objective is to provide a more efficient operation through remote job entry. However, if the objective is to establish an environment in which the computer is part of a continuous thought process, the improvement in response time from days to minutes is not sufficient because the human mind requires response times in the order of seconds for continuous thinking. Hence, the following observed characteristics of the mind play an important part in the design of a computing system.

Short Term Memory--When tasks are performed, a body of information is held in the mind at conscious level, termed "short term memory" by Miller (ref. 6). Two characteristics of short term memory, both associated with waiting, are important.

- a) Short term memory is never passive. Noise from within the mind or distractions from without can cause change of its contents. The risk of loss of information rapidly increases when a person is conscious of waiting. Consciousness of waiting occurs within two seconds after closure (see below) if new activity is not begun.
- b) During creative or highly innovative periods, large amounts of work are performed within continuous, concentrated, and relatively short time periods. Interruptions of less than a minute during one of these periods can cause loss of the entire line of thought.

Closure--Humans spontaneously organize their activities into "clumps" (ref. 6) that represent an action that is concluded with a definite result. An example, is looking up a number in the telephone book followed by dialing the number. At the end of each of these activities there is a sense of completion. Psychologists call this sense of completion a "closure."

A closure is also the point where the minimum information necessary to proceed to the next clump is held in short term memory. Hence, interruption of a clump of activity results in a closure and a partial purging of short term memory to only that information necessary to handle the interruption. This is observable when dialing the telephone where an interruption will cause loss of memory of the number being dialed and, if the interruption is intense enough, loss of memory that the phone was being dialed.

When solving complex problems, short term memory is heavily filled. The ability of a person to solve complex problems is directly related to the amount of information he can hold in short term memory and the concentration with which he can achieve a chain of closures leading from one conclusion to the next. Interruption of this process nearly always results in a "restart" and, as stated above, can result in loss of the entire activity.

Closures come in different degrees depending upon the importance of the result. A person is much more tolerable to interruption when an important closure has been reached than he is at an intermediate closure.

Step-Down Discontinuities--The rate at which thought processes decrease in efficiency as the number and length of response delays increase is not continuous. For example, intense creative dialogue is not possible with response times greater than 2 to 4 seconds. Ordinary conversational dialogue becomes awkward with response times greater than 2 to 4 seconds and is not possible with response times in excess of 15 seconds. When two persons are holding the dialogue, the response need only be a nod or a grunt but it must occur within the given time period to avoid feelings of anxiety or a breakoff of communication.

Where a continuous thought process involving the computer is not a necessary part of the problem solving activity the user is engaged in, the above observations and the paragraph on response time are not relevant. But it should be noted that the user will not engage the computer for some types of activity unless he can do so at conversational speeds in a mode that is compatible with his thought processes. Useful tasks will still be completed when these criteria are not met but some of them will be done less well. Direct use of the computer for problem solving, creative processes, and complex interrogation requires a conversational man-computer dialogue.

6.4.2 Response Times

The response times given are those for which the user will be comfortable and continue to utilize the terminal for his purposes. They are a quantitative expression of a qualitative phenomenon and, as such, are subject to interpretation. However, they are based upon study and observation, and, while the association between response time and activity may not be precise, such an association does exist and is of the order given. Response time is defined as the time elapsed between the last input by the user and the first character displayed by the computer.

Classifications--The following relationships between response time and activities are extracted from Miller (ref. 6) and Martin (ref. 7). They are illustrated in figure 6.9.

- | | | |
|--------------------|---|--|
| >1 minute | - | Essentially no interactive activity. |
| >15 seconds
but | - | (1) Some log-on/log-off functions where the user is familiar with the delay. |
| <1 minute | | (2) Single enquiries where the user is familiar with the delay, preferably cued by a message from the computer |

within 2 seconds acknowledging the command.

- (3) System failures and recoveries, preferably cued, where possible, by a message from the computer within 2 seconds warning of the delay.
 - (4) Loading of programs and data for execution and processing, preferably cued by a message within 2 seconds acknowledging the command.
 - (5) Restart from yesterday.
 - (6) Conversational dialogue is not possible.
- >4 seconds but - (1) Low key enquiry dialogue possible but awkward.
- <15 seconds (2) Intense creative dialogue not possible.
- >2 seconds but - (1) Complex enquiries where continuity of thought is necessary.
- <4 seconds (2) Initial acknowledgment by the system that it is "listening."
- (3) Error messages.
- <2 seconds - (1) Intense creative dialog .
- (2) Acknowledgment by the system that a command has been received.
- (3) Response to a paging request through a keyboard.
- <1 second - (1) Response to a paging request using a light pen.
- <0.1 second - (1) Brightening of characters from a light pen selection.
- (2) Appearance of a line when using the light pen as a drawing stylus.
- (3) Appearance of a character on a CRT keyboard.

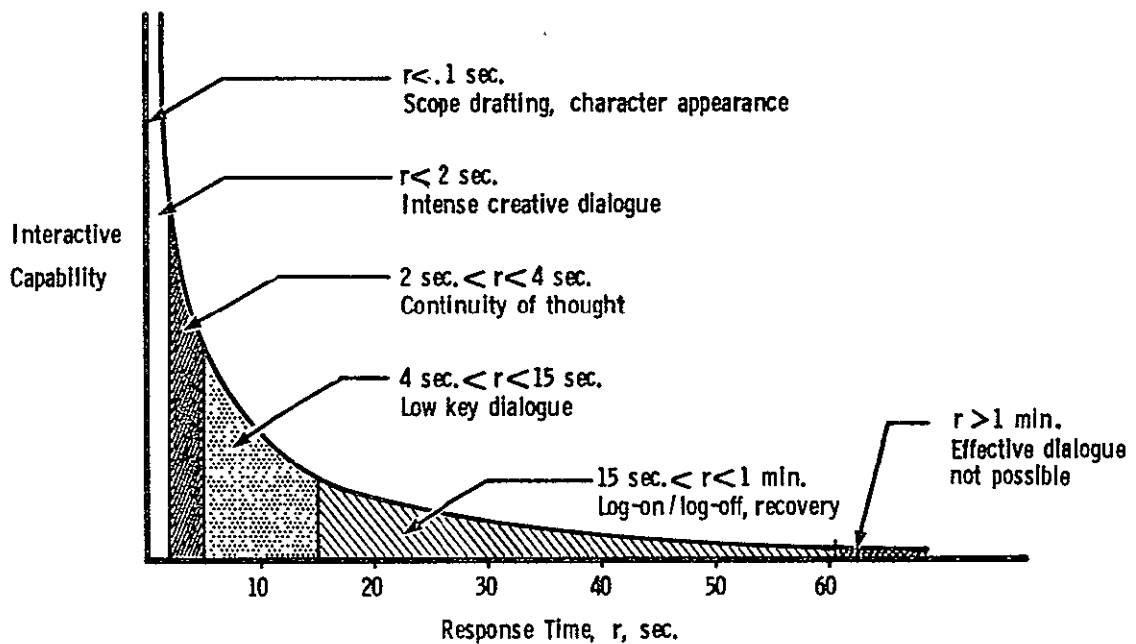


Figure 6.9. Interactive Response Time

Miller (ref. 6) recommends that error messages be delayed for 2 seconds and displayed within 4 seconds. This delay allows the user to reach closure before he is faced with a need to redirect his thought processes to correct errors. Instantaneous error messages or error messages that interrupt the user in mid-command are disruptive and cause confusion and frustration. This is more true for the casual user than for the dedicated user.

The critical threshold for effective creative dialogue is 2 seconds. Beyond 2 seconds mental efficiency degrades rapidly. Delays beyond 15 seconds should be structured to relieve the user of both mental and physical captivity. Experienced users will prefer faster response times.

Deviations--Permissible deviations in response times vary. In general, the permissible deviation depends upon the seriousness

of the closure to the user. Response times in the 2 second and less category should not vary by more than 100%. The curves in figure 6.10 from Martin (ref. 7) are illustrative of good and bad response time deviation characteristics.

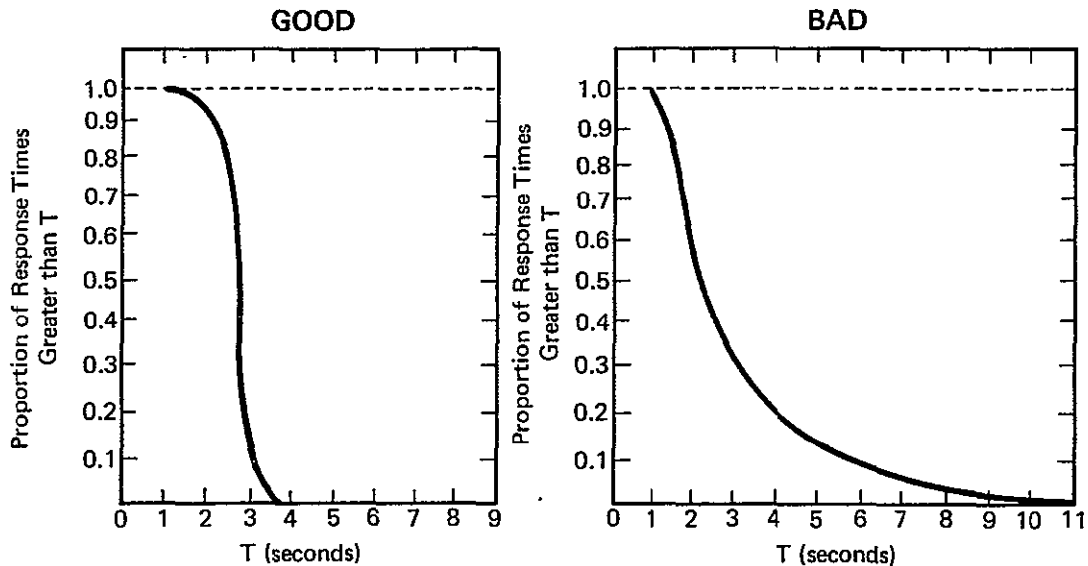


Figure 6.10 Response Time Deviations

6.4.3 User Classifications

Familiarity--The complexity of each problem step a user is able to handle decreases proportionately, if, through unfamiliarity, a user's short term memory is filled with personal concern or memorized step-by-step procedures. Hence, the unfamiliar user must be helped by decreasing the complexity of each step, increasing the number of steps, and increasing the volume of reminder information supplied. The reverse is true when the user is familiar with the activity.

Expectation--Responses strongly dissimilar to the user's expectations are the same as an interruption. Less obvious and

less critical, but still important, is the style of the language. Quick cryptic language statements may appear coarse and rude to the manager who's day-by-day business requires close attention to a smooth interface with people. On the other hand, language that is polite and uses full English may be boring and time consuming to the technical specialist. As Martin (ref. 7) says, dialogue design must "steer a course between operator boredom and bewilderment."

Classifications--The above factors lead to the following classification of users.

a) Totally Untrained or Novice--This user:

- is likely to be intimidated by the terminal,
- is consciously defensive,
- has his short term memory almost completely filled with information related to learning and very little to problem solving, and
- is easily frustrated by unclear terminal responses or unusual response times.

He requires:

- programmed learning,
- tutorial dialogue,
- minimum opportunity for error, and
- display messages that maximize his confidence in himself and in the system.

b) Casual--This user:

- spends most of his time doing something other than operating a computer terminal,
- is trained in terminal usage and feels at ease using it,
- remembers general procedures but forgets specific commands and formats, and
- expects to return to a system not grossly different from his last use.

He requires:

- optional tutorial dialogue,
- descriptive cues and prompts to remind him of missing information, errors in command structure, etc.,
- minimized use of mnemonics,
- insignificant change to syntax and sequences between uses, and
- small deviation in response times.

c) Dedicated--This user:

- spends most of his time operating a computer terminal,
- has near instantaneous recall of command structure,
- is psychologically tuned to the response pattern of the terminal,
- is adaptive to command structure changes, and
- is intolerant to language structure beyond the minimum required for uniqueness.

He requires:

- abbreviated cues and prompts,
- maximized use of mnemonics, and
- faster than normal response times.

6.4.4 Man-Machine Dialogue

The system is visible to the user only through the command structure. The command structure of the dialogue is related to the system in the same way a person's speaking habits are related to the person. The users expectations will follow directly from the class of user he is (as defined in the previous section) and his personal nonprofessional characteristics. The users psychological acceptance or rejection of the dialogue will be based upon how well the system

capabilities, language structure, and response times match his expectations.

In general, the man-machine dialogue should have the following characteristics:

- a) The dialogue should be compatible with the way the task is organized, i.e., the dialogue should be flexible where task organizations are variable.
- b) The extent of the computer responses and those of the user should be compatible with the user's training and experience.
- c) Completion of an activity should be punctuated by a closing act in the dialogue.
- d) Signals should be given when the computer is listening, both immediate and interim when the computer activity is long.
- e) Where groups of associated data are being input through a dialogue, the computer should "clean-up" and appropriately display the data at convenient times.
- f) The structure of the dialogue should minimize errors at input.

Specific classes of dialogue are discussed below.

User_ Initiated--User initiated dialogue implies a dedicated user, or at least a user of such frequency that the dialogue commands are instantaneously recalled. Classes of user initiated dialogue are given below. It should be noted that the user is leading and the computer is interpreting and responding.

- a) Full English--Because of the alternate meanings of words and the context dependency of English statements, interpretation of full English syntax by the computer is difficult. Further, full English is responsive to characteristics of the human mind that are not present when the computer is a party to the dialogue. Hence, full English is not a viable communication language for man-machine dialogue.
- b) Limited English Input--English words and phrases can be used where distinctive meanings can be assigned. However, use of full English statements where some of the words are read by the computer and the rest are ignored is often confusing to the user. For example,

PLEASE DISPLAY ALL BEAM ELEMENT NAMES AND
MARGINS OF SAFETY WHERE THE MARGIN OF
SAFETY IS GREATER THAN 0.95.

In this example, the underlined words are the only words interpreted by the computer. In this type of dialogue the user must know the precise words the computer will read and their required order. Misspelled command words will, of course, be ignored. The possibility for misinterpretation is great. The temptation to try a series of words without determining the command words is also great. Hence, this type of "mixed" dialogue should be limited. If used, the actual command read by the computer should be displayed back. The above command would be better given as

DISPLAY BEAM ELEMENT NAMES AND MARGINS
WHERE MARGINS GT 0.95.

In this command, every word is read and has meaning to the computer and the entire phrase has meaning to the user. This type of language (i.e., limited English without extraneous words, user initiated) is probably the most useful language form for the casual user of IPAD.

- c) Mnemonics--Mnemonics is the most efficient language for the dedicated user. They given great flexibility and forego extraneous characters not required to uniquely identify the command to the computer. The disadvantage is that they must be remembered and present little in the form of memory aid. The above command might appear in mnemonics as

D / 2 B ID, M / * M GT 0.95.

- d) Graphic--This dialogue is initiated by the user drawing lines, shapes, or symbols; or graphically supplying instructions to change the size or location of the same either through the CPT face or via an electronic tablet.
- e) Pictorial--This dialogue is initiated by the user by manually or optically tracing and marking a drawing to be stored in the computer. It may also be initiated through calls for display of stored picture catalogs with corresponding commands for paging and selection.

Computer Initiated--Computer initiated dialogue is necessary where the user is unfamiliar with the command structure or data input formats and must be directed or "led through" the procedure. It should be noted that the computer is leading and the user is following.

- a) Full/Limited English--Full English commands with limited English or mnemonic user responses is the most appropriate dialogue where the user is entirely unfamiliar with the procedure. Menus, lists of alternatives, explanations, and helps are all a form of this command.
- b) Mnemonic--Where the user is entirely familiar with the mnemonic set but unfamiliar with order of input, a computer initiated dialogue using mnemonics can be used.
- c) Form Filling--A form can be displayed giving appropriate blanks and headings.

Hybrid--Combinations of user initiated and computer initiated dialogue can be useful.

6.4.5 Errors and Failures

Effect of Language--The language response must be consistent with the user's mode. If the user is making a single inquiry, he will probably have note of it, and a request for reentry is adequate. If the user is making a complex inquiry, it will be necessary to display an index of categories or parameters he previously input to place him back in context. If the user is in a conversational problem solving mode, the data he has constructed to the point of error or failure must be available to him. Reconstructing data is one of the most arduous and unreliable activities he performs. Loss of a batch job means only that the job must be rerun. Loss of a creative terminal job means the model must be reconstructed. Reconstruction is a demoralizing activity. Hence, error correction or restart after system failure must be responsive to the user's need to (1) retain confidence in the work thus far completed and (2) retain confidence in the terminal as a problem solving medium.

Diversity of Source--The integrity of a data set will generally be less when multiple independent users are inputting to it than when the data is received from a single controlled source. Hence, procedures for achieving integrity of the contents of data sets increase in importance in a multiple user community. The following procedure is recommended.

- a) Real time dialogue to detect and correct errors.
- b) Software for performing scans, sums, cross file checks, etc., of the entire data set.
- c) Intelligent methods of correcting errors and the effects of errors discovered at a time subsequent to input and first usage.
- d) Designation of ownership responsibility to some member or manager of the user community.

Interruptions--Interruptions have been discussed in the previous sections. To summarize, interruption of the user to inform him of errors or to warn him of impending system failure should occur, if possible, at closure rather than during activity.

6.4.6 System Balance

Human factors must be balanced against other factors such as cost, hardware capabilities, etc. For example, a dialogue that has voluminous computer responses and mnemonic user responses overbalances line usage in one direction, which, a) affects response time, b) reduces the number of terminals that can be multiplexed on a single long line, and c) increases the cost of the system. In this instance it may be necessary to shorten the computer responses, or store the responses locally and trigger them with mnemonic signals from the computer. In summary, factors such as transaction time, number of terminals, line costs, and human effectiveness must be carefully balanced to achieve the most cost effective system.

6.5 SECURITY

A secure system is a design goal of IPAD. Accessable items will be protected from unauthorized access and use, by a system of passwords, answerbacks, security classifications, clearances, etc. Accesses will be logged so that attempted security violations may be determined through security audits.

6.5.1 Definitions

Security Classification (SC)--The security classification of an entity is the total set of all codes, flags, passwords, answerbacks, algorithms, access modes, etc., assigned to that entity for the purpose of control. Entities α consist of:

- IPAD System
- User
- IPAD System Commands
- Project
- Subtask
- Library Entries

The security classification assigned an entity will be used to control log-on and access as well as functions performed after obtaining access. This classification will be denoted SC_{α} for entity α .

Security Clearance (C)--A security clearance is the security classification for a user. It will consist of the following items:

- User ID
- Password
- Government or Company assigned clearance, i.e., CONFIDENTIAL, SECRET, etc.
- Allowed operations on specified entities

Required Log-on or Access Sequence--Log-on will consist of the sequence of input items required of a user. This may or may not be an enforced order sequence, i.e., constitute an ordered set. Usually there will be an acceptable order that might be imposed. Hence, it will be assumed to be an ordered sequence unless specified otherwise. Each entity will require a log-on sequence denoted by $\alpha_1, \alpha_2, \dots, \alpha_n$. Note that the α_i 's will be functions of a user u so he must furnish the sequence $\alpha_1(u), \alpha_2(u), \dots, \alpha_n(u)$ in order to log-on ($\alpha = \text{user}$) or access entity α ($\alpha \neq \text{user}$). For example:

$\alpha_1(u)$ = user User ID
 $\alpha_2(u)$ = user Password
 $\alpha_3(u)$ = user Clearance - CONFIDENTIAL, SECRET, etc.

•
•
•

denote the required α log-on sequence for user u by $L_{\alpha}(u)$.

User Security Profile (SP)--For a given user u with log-on or access sequence $L_{\alpha}(u)$ for entity α , certain rights and constraints will be granted and imposed subsequent to a successful sequence input. These rights and constraints will constitute his implied security clearance $I_{\alpha}(u)$. This clearance, together with the required input sequence, will constitute the user's security profile for α , denoted $C_{\alpha}(u)$. Hence,

$$C_{\alpha}(u) = L_{\alpha}(u) \cup I_{\alpha}(u).$$

$C_{\alpha}(u)$ will also be called u 's security clearance to access α . It should be noted that $C_{\alpha}(u)$ must be contained in SC_{α} , i.e.,

$$C_{\alpha}(u) \subseteq SC_{\alpha}.$$

The totality of all such user profiles for each α will be called the user's security profile $SP(u)$. Hence,

$$SP(u) = \bigcup_{\alpha} C_{\alpha}(u).$$

Potential Security Violation--A user u is required to furnish an input sequence $L_{\alpha}(u)$, if he is to be validated for log-on or access to entity α . Let $L'_{\alpha}(u)$ denote the input sequence actually supplied by u and $I'_{\alpha}(u)$ the resulting implied clearance allowed. Let

$$C'_{\alpha}(u) = L'_{\alpha}(u) \cup I'_{\alpha}(u),$$

then if

$$C'_{\alpha}(u) \not\subseteq C_{\alpha}(u),$$

a potential security violation is said to have occurred. Thus, if a potential security violation (also called potential threat) occurs, then $L'_{\alpha}(u) \not\subseteq L_{\alpha}(u)$. The reason for choosing "not a subset" as opposed to "not equal" is that a partial input sequence $L'_{\alpha}(u) \subseteq L_{\alpha}(u)$ might be supplied by the user resulting in the partial implied clearance $I'_{\alpha}(u) \subseteq I_{\alpha}(u)$. Hence,

$$C'_{\alpha}(u) \subseteq C_{\alpha}(u).$$

This would allow the user some but not necessarily all the access freedoms available.

Access and Security Logs--Two types of logs will be defined: an access and a security log. Each time a user accesses an entity an entry will be made in the access log. This entry will consist of the triple $(u, \bar{\alpha}, \text{date-time})$. Here, $\bar{\alpha}$ denotes information about the access to α . The access log may exist

both as an explicit log used, for example, during IPAD log-on, or an implicit log in the case of a data set. In the latter case, access information will be saved in the library directory entry.

The security log will be used to log access information of a more sensitive nature. In particular, potential security violations will be entered in this log. Entries will consist of the quadruple $(u, \bar{\alpha}, v, \text{date-time})$ where $\bar{\alpha}$ denotes information about access to α , and v denotes the nature of the security associated with the access to α . This could include security needed and received, type of security threat, which occurrence of consecutive threats, severity of threat, etc.

Security Audit--A security audit will be performed periodically on the security log to determine attempted security violations. This audit will be available in greater or lesser detail, on option, to the responsible users and managers. This audit may be obtained by request or occur automatically, possibly triggered by the severity of some potential threat.

6.5.2 IPAD Security Initialization

IPAD Security File--The IPAD security file will contain the security profiles of all valid users as well as the security classifications for the IPAD system, projects, subtasks, and library entries. The file will consist of an explicit part and an implicit part. The implicit portion of the IPAD security file will reside in the library directory entries of the various entities and the explicit portion will be contained in an actual security file. Unless context specifies otherwise the IPAD security file will mean the explicit part.

The IPAD security file will reside as a proprietary file in the IPAD data base. It will be initialized with a user's security profile and α security classification for $\alpha = \text{IPAD system}$. This will constitute the log-on sequence required together with the rights of a given user. Each user, to log-on to IPAD, must have an IPAD security file entry.

Initialization of the security file with a user profile will not be allowed in the normal mode of IPAD. This must be done under management control or outside of IPAD. For example, a special batch program will be required or only one special terminal will be allowed to create and update the security file. The initial entries required for a user profile as regards log-on sequence are as follows:

- User ID--An identifier assigned a specific user.
- Password--A password assigned a user.
- Answerback (Optional)
 - Last name
 - Mother's maiden name
 - Social Security number
 - etc.

Optional entries needed to complete his profile are, for example:

- User may create and enter project definitions.
- User may cancel a project.
- User may update another user's profile.
- User's profile may not be altered by anyone without user's status x.
- User's status
- etc.

The above profile entries constitute the user's profile for access to IPAD denoted $C_I(u)$.

Project Security Information--Project security information will be established as an implicit part of the IPAD security file. It will be initialized by someone validated by the IPAD security file to enter and define projects. The project plan will contain a project security profile for each user associated with the project and additional security classifications required by the project. Entries in the project plan for a given user may be:

- User ID (input supplied by IPAD log-on)
- Answerbacks, etc.
- User can enter and define project plans
- User may update a project plan.

- Project related items.

The above entries will constitute the user-profile for access to a project, denoted $C_p(u)$. Note that $C_I(u)$ must also be defined.

Subtask Security File--The subtask security file is derived from the project security information. Entries in this file may include the following:

- Subtask Password
- User may define data sets
- User may purge data sets
- User may execute specific subtask related commands
- User may change his subtask password
- Subtask related items

The subtask profile for a given user is denoted $C_{ST}(u)$.

Library Entry Security Information--This information will consist of the user library entry profiles together with library entry security classification. Items included in this set for a given user may be:

- Library Entry Password
- User ID
- Access Mode Allowed
 - Read
 - Execute
 - Extend
 - Modify
 - Purge
- Clearance required, CONFIDENTIAL, SECRET, etc.
- Accessable time or dates
- Library Entry related items

The library entry profile for a user will be denoted $C_{LE}(u)$.

User Security Profile--The totality of all profile for all entities constitute the user security profile $SP(u)$ which is equivalent to his clearance $C(u)$. Moreover, it also consists of all access input sequences required, together with clearances granted on successful input sequence submission. Hence,

$$C(u) = SP(u) = \bigcup_{\alpha} C_{\alpha}(u) = \bigcup_{\alpha} [L_{\alpha}(u) \cup I_{\alpha}(u)].$$

The last equation may be translated into the following matrix:

$C(u)$	α	$C(u)$			
	IPAD System	User ID	Password	Answerback	Projects* etc.
	Project	User IDs	Password	Answerback	etc.
	Subtask	User IDs	Password	Answerback	Library Entries* etc.
	Library Entries	User IDs	Password		Access mode etc.
	etc.				

* user is allowed to define these entities.

The above matrix, in skeleton form except for entries needed to log-on to IPAD, will be entered into the IPAD security file when a user's profile is initialized. Subsequent matrix entries will be made by those validated as project, subtasks, and library entries are developed. This matrix will be available in the IPAD system for security checking when access to various items or function is requested by the user.

6.5.3 Accesses and Requests

IPAD Log-On--The user must first satisfy the host operating system log-on protocol. Having done so, he will enter IPAD by supplying his log-on sequence, denoted $L'_{\alpha}(u)$. This will consist of:

- User ID
- Password
- Subtask Identifier

- Answerbacks, etc.

Once his input sequence is complete,

$$L'_\alpha(u) \subseteq L_\alpha(u) \text{ and } I'_\alpha(u) \subseteq I_\alpha(u),$$

implying the user's stated clearance satisfies IPAD's security classification for that user, he may proceed to log-on for his subtask. The user will be primarily entering and accessing library entries. His right to do so and in what mode will be contained in his security profile established to date.

IPAD System Requests--Certain IPAD functions are categorized as security classifiable. Examples are:

- DEFINE
- ENTER
- DISPOSE
- MODIFY
- DISPLAY
- SEARCH

Each user's profile will contain permission codes controlling use of these functions. For example, only specific users will be allowed to send data to a remote location or purge a library entry from the community library.

Suspending or Revoking Clearances--At any time, due to a security alert, change of project plan, etc., a blanket revocation of log-on or access may be imposed. This revocation may be made by any one authorized to alter security profiles. The revocation will be reflected in all appropriate user profiles when entered. An affected user's progress will be suspended the next time his security profile is checked by the IPAD system.

Potential Security Violations--If during log-on, access or requesting an IPAD function, a user enters an input sequence $L'_\alpha(u)$ not contained in the required sequence $L_\alpha(u)$, a potential security threat exists. This threat will be logged in the security log and a threat count started. Depending on the severity of the violation; sensitivity of project, subtask, or library entry, the threat count will trip a security alert when it reaches a certain value. Again, depending on the severity of the threat, action will be taken. This action may range from

requesting an additional password, certain answerbacks, telephone, or even manual verification. If the threat is not severe, the user may be logged out. If the threat is severe, a security alert may be issued by automatic notification of a security office.

6.5.4 Privacy and Integrity

Privacy--Privacy is the right to keep information private to oneself and the guarantee that such information will be kept safe from unauthorized access. The question of whether such information may be obtained and kept is a legal and administration question. The IPAD software design allows varying degrees of privacy depending on how the security controls are used.

Integrity of Content--A checksum will be made of information entered into the IPAD data base. This checksum will be updated whenever the information is altered. A user may request a checksum verification, at any time, to determine if a recomputed checksum compares with an alleged checksum. To insure that a checksum itself has not lost integrity, the checksum will be kept with an additional check digit. The checksum verification will ensure that, after entering information into the data base and determining it to be correct, by visual read out, comparing, displaying, etc., any loss of integrity can be detected by the user.

6.6 STANDARDS

Webster's Dictionary defines standard as, "That which is established by authority, custom, or general consent, as a model or example; criterion; test."

Standards are necessary for an orderly working world. Without them chaos would reign supreme; communication would be ineffective or at best, very difficult; time would be wasted; and learning would be severely affected.

Standards are not to be had without a price. This price is paid in work in defining and establishing standards and in learning and observing them. Standards have advantages and disadvantages. Some disadvantages are:

- Standards are inflexible
- Standards stifle creativity

However, standards must be inflexible to bring order to work. Some also argue that standards stifle creativity and thereby confuse creativity with the application or observance of standards. The real difficulty with standards are:

- Standards must be decided upon
- Standards must be learned and observed

While most people agree that there should be standards, it is usually difficult for them to agree on what standards should be.

Standards are necessary for the orderly maintenance of activity. They facilitate effective communication, reduce the effort required to learn, and insure the effective application of education. Standards enhance reliability, consistency, and integrity by reducing errors and mistakes. In summary, their advantages are:

- Facilitate communication, teaching, and learning
- Save time
- Insure reliability, consistency, and integrity
- Minimize errors and mistakes
- Inflexibility

When establishing standards an attempt should be made to optimize their definition and development, so as to maximize their advantages and minimize their disadvantages.

6.6.1 IPAD Design Standards

Standards have been adopted in the design of IPAD. These standards are given as follows:

Structured Top Down Design--Top down design means starting with the most general view of IPAD as seen by the user community, and dividing it into constituent parts. Each refinement comprises another level in the design. Any one level may be thought of as describing what the design at that level consists of, with the next level below giving the how of the preceding level.

Independent Modules--The design results in a set of program modules that may be implemented and modified as independently as possible from other modules. This is a design standard, and

fortunately a consequence of the structured top down design process.

Open Ended--It is envisioned that IPAD will undergo continual development. To accommodate this development and expansion, open endedness is a design standard.

Machine Independence--IPAD as a design is to be independent of specific vendor hardware.

Data Base Management--To facilitate I/O, a data base management system is a basic design standard of IPAD. All standard I/O in IPAD will be through the data base management system.

6.6.2 IPAD Implementation Standards

Standards adopted for IPAD implementation should be consistent with those established in the design. Implementation standards should be chosen to facilitate program maintenance and checkout. The code should be written neatly, uncluttered, and readable; it should be written for the novice and not for the coder.

The list of items given below fulfills the above needs.

Language--A common higher level machine independent language should be adopted. It should be the implementation language for IPAD in which most of IPAD would be written.

Modular and Open Ended--As code is developed, modularity and open endedness should be kept in mind. They are IPAD design standards.

Common Code--Areas in IPAD whose function can be served by one common block of code should be identified. This will guarantee consistency of function; coded, checked out, and performed in one place as opposed to many.

Program Blocks--Blocks of program code should be structured in an overall sense much like a book or document. The program block should contain the following items as a minimum:

- Title Section--author, date, etc.
- Revision Section--modification and revision history
- Abstract Section--stating the purpose, method, etc., for the program block

- Bibliography Section--contains any external references related to the program block
- Usage, Input/Output, etc.--sections describing use of the program block, input required, output generated, and other such related items.
- Quality Assurance Section--this would contain a description and history of required steps and actions needed to checkout and certify this block.
- Certification Section--this would contain names and references of who modified, tested, and approved the block for release.
- Program Section--this would be that portion of the block comprising the executable statements, arrays, variables, formats, etc. It may be further subdivided into structured sections.

Minimize Host System Interface--One critical design goal of IPAD is to minimize the host system interface. In the structured top down design, the actual host system IPAD interface will be delayed to the lowest level possible. Standards will be developed for IPAD/Host system interfaces.

Documentation--The entire program documentation should be structured to facilitate programmed extraction for production of a particular program document.

Naming Conventions--A consistent convention should be adopted for naming and distinguishing variables, arrays, tables, constants, code block names, etc. These names should be short, 3 or 4 characters as opposed to long, 6 or 7 characters. Those items related to general system activities should be identified, named, and used throughout in a consistent manner.

Coding and Documentation Conventions--Program coding should adhere to the framework of established program structure. Executable program statements should be distinguished from documentary statements. They may, for example, be offset by blank lines and indented, using a hierarchal statement convention.

The executable code program logic should be well documented. This documentation should be meaningful and uncluttered. Comments should be informative and not merely restate executable statements. Liberal use of outline conventions, blank lines, and blank spaces should be used. Identifying and setting off items with non-blank special

characters should be avoided at all costs. Embellishments add clutter and decrease readability.

- Variables should always be used with the values initialized in one place. Constants should be used with discretion.
- Labels should be lexicographically ordered, increasing in the direction code is read.

Certification--Certification is defined to include checkout, approval, and release of a block of code.

As part of the development of a block of code, a description of the checkout necessary should be entered in the quality assurance section of the program block. A checkout procedure, test data, and code should be assembled and placed in a quality assurance library. This will then be used to certify the block. Individuals responsible for checkout and approval will be recorded in the quality assurance section. The revision section will be updated to reflect the change if meaningful. The block will be checked out, approved, and released with the appropriate entries having been made in the certification section. This process will constitute certification.

6.6.3 IPAD Maintenance Standards

IPAD maintenance can include both ongoing development as well as modification and maintenance of existing code. All code should be developed using the same standards created for IPAD implementation.

All development and modifications should be documented as established by the standard. This would include both program code annotation as well as updating the modification record sections. Development, per se, is to be distinguished from error correction.

Modifications should be described whether resulting from development or errors, and the verification procedure documented in the quality assurance section. The program code should then be tested and certified in the usual manner to insure it will work. The checkout procedure, decks, test data, and documentation for the current modification should be added to the quality assurance library. IPAD should then be re-certified for release.

6.6.4 IPAD Application Standards

Standards related to the IPAD user interface will be determined primarily by implementation. For example, the IPAD standard for packaging computing and operational modules will be determined when the actual IPAD command language is defined, the implementation language known, data base established, and the host system hardware configured. Standards related to the IPAD user community will also be determined by implementation. These standards will deal specifically with a particular IPAD implementation and must be user initiated.

One area related to the IPAD user community and standards, however, needs further study. It is an area that greatly impacts all IPAD users and should be considered as becoming an IPAD standard. This area deals with the following items:

- Dimensional Units--The metric system of units (MKS) might well be taken as an IPAD standard.
- Constants--Numerical constants such as π , e , etc., should be standardized with respect to nomenclature and significance.
- Physical Constants--Constants such as: speed of sound, gas constant, gravitational constant, etc., should be identified and standardized as to units, nomenclature, and significance.
- Physical Variables--The terms: velocity, acceleration, mass, force, etc., should be identified and standardized as to units and nomenclature.
- Miscellaneous Terms, Abbreviations, and Symbols--Terms such as Mach number, lift, planform; abbreviations such as a.m., p.m., hr.; and symbols such as ∇ , Σ , \int , etc., should be identified and standardized.
- Disciplines--Engineering and design process disciplines, such as structures, loads, trade studies, etc., should be defined and standardized. This could include a standard for planforms, a global airplane coordinate system, substructure coordinate systems, and the like.

6.7 LANGUAGE REQUIREMENTS

A substantial number of computer programs currently exist that are candidates for inclusion as application modules in IPAD. These programs are predominately FORTRAN but other languages are represented. Although FORTRAN is a universally

accepted language, many dialects exist. Additionally, FORTRAN contains machine dependent characteristics requiring a specific combination of source language statements, compiler, operating system and computer hardware.

It is a design requirement that IPAD be capable of accepting pre-existing application modules. It is also expected that IPAD host systems will be on third and fourth generation hardware of more than one manufacturer. Further, it is very desirable that IPAD have languages for all user functions that are independent of the host system. That is, all IPAD functions at the user interface should not vary with changes of the host system.

A practical way of handling the investment of the aerospace industry in existing FORTRAN programs must be developed initially. For the long term, IPAD should accept other programming languages such as ALGOL, COBOL, APL and PL/I. The study made by Control Data Corporation, consultant to Boeing, considered:

- a) The general problem of software migration;
- b) FORTRAN source code migration on third generation computers;
- c) Migration from third generation to fourth generation computers;
- d) The development of a machine independent FORTRAN.

The final report of this study is given in Appendix C. A recommendation is made in the study for the development of a machine independent FORTRAN language, IPADF, that could be used for the implementation of the IPAD system. It is also recommended in the study that utilities be developed for translating existing FORTRAN application modules into IPADF. The need for a machine independent language may be even more general than recommended by this study. Such a language is referred to elsewhere in this volume as IPADL.

Languages are also required at the user interface of IPAD. One user interface is at the host operating system level through languages such as OS360/370, JCL or CDC 6600 SCOPE or KRONOS control statements. The language associated with the IPAD commands and utilities must be specified. Human engineering factors are given in section 6.4 but the syntactic forms remain to be developed.

7.0 HOST SYSTEM SPECIFICATIONS

This host system specification considers current and future hardware and operating system software. Information was obtained from several manufacturers of large scale computer systems detailing their products. Two sample host system configurations have been given based upon:

- a) a Control Data 6600 (Cyber 74) and
- b) an IBM 370/168.

These computers were chosen for presentation because of their widespread use in the aerospace industry. They are illustrative and are not intended to be a recommendation of these particular manufacturers at the exclusion of others.

7.1 VENDOR SURVEY

To maximize portability in the IPAD system design, and to insure that all potential hardware was considered in the design of the system, all manufacturers of large scale computer systems were surveyed. The survey covered the following areas:

- a) Mainframe
 - o System architecture, (multiprocessor, etc.)
 - o Instruction type, complexity, timing, and rate
 - o Character, integer, and floating point representation
 - o Main memory size/access rate
 - o Input/output rates and number of channels
 - o Multi-programming capability
 - o Time-sharing capability
 - o Reliability
 - o Member of a compatible family of computers
 - o Cost
 - o Public availability date
- b) High Speed Random Access Storage
 - o Capacity
 - o Transfer rates
 - o Latency
 - o Dismountability
 - o Cost
 - o Public availability date

- c) Mass Storage (\approx trillion bits)
 - o Capacity
 - o Transfer rates
 - o Number of ports
 - o Recording media
 - o Dismountability
 - o Cost
 - o Public availability date
- d) Data Transmission Peripherals
 - o Rates
 - o Capacity
 - o Cost
 - o Public availability date

The questionnaire was mailed to the following vendors:

Burroughs Corporation
 Paoli, Pennsylvania
 Control Data Corporation
 Minneapolis, Minnesota
 Honeywell Information Systems, Inc.
 Waltham, Massachusetts
 IBM Corporation
 White Plains, New York
 Sperry Rand Corporation, Univac Division
 Washington, D.C.
 Texas Instruments, Inc.
 Austin, Texas.

Replies were received from all vendors contacted. They were understandably reluctant to divulge future plans but were quite willing to supply detailed performance specifications of their presently marketed systems. The hardware characteristics of the submitted mainframes and peripherals of all vendors satisfy the IPAD system requirements. Instruction rate, or CPU power, of some systems is sufficient for a small-to-medium scale installation. In some cases, a medium scale installation would dictate a multiprocessor configuration. A large IPAD installation (for example, one capable of supporting the design of a supersonic transport as outlined in Volume II) would require at least one CDC 7600, IBM 370/195, or Texas Instruments ASC central processor. There will be a heavy demand upon the timesharing, data storage, and data handling capabilities of these systems. The information presented is accurate as of late 1972. Table 7.1 is a condensed comparison of these computer systems.

	APPROXIMATE INST. RATE MILLIONS PER SEC.	MEMORY SIZE/RATE MILLION CHAR. MCHAR/SEC.	I/O RATE # CHANNELS MCHAR/SEC.	CHARACTER SIZE F.P. PRECISION BITS, BITS	COMMENTS
BURROUGHS B7700	12-18 EACH*	7-8 12	32/IOP 8/IOP	6,8 40	HIGHLY MODULAR MULTIPROCESSOR WITH VIRTUAL MEMORY
CDC CYBER 74 (6600)	3-5	.6-1.3 100	12 8-10	6 48,96	OVERLAPPED SCIENTIFIC INSTRUCTION PROCESSOR
CDC CYBER 76 (7600)	20-25	1.6-5.7 360	15 25-50	6 48,96	OVERLAPPED SCIENTIFIC INSTRUCTION PROCESSOR
HONEYWELL 6070/6080	1.2 EACH	1-6 40-90	24 6/IOM(1-4)	6,9 28,64	GENERAL PURPOSE MULTIPROCESSOR (1-4), 6080 HAS MANY CHAR. INSTR.
IBM 370/165	4-7	.5-3 16	7-12 1.3-3	8 BIT EBCDIC 21,53,109	THE 370/168 HAS VIRTUAL MEMORY, BOTH HAVE 80 NS. CPU BUFFER STORAGE
IBM 370/195	15-18	.5-4 170	7-12 1.3-3	8 BIT EBCDIC 21,53,109	ELABORATE OVERLAP PLUS 54 NS. CPU BUFFER STORAGE
TI ASC	30-50 **	4-16 -3200	4-12 28	8 21,53	1-4 PIPELINE VECTOR/MATRIX PROCESSOR WITH VIRTUAL MEMORY
UNIVAC 1110	1.8 EACH	7.8 12	8-24 24 TOTAL	6,8,9 27,60	GENERAL PURPOSE MULTIPROCESSOR (2-4)
* Each processor of a multiprocessor machine ** Arithmetic only. Does not include fetch, store, index, and branch operations.					

Table 7.1 Comparison of Vendor Hardware

ORIGINAL PAGE IS
OF POOR QUALITY

7.2 HARDWARE CHARACTERISTICS AND CAPACITY REQUIREMENTS

All available large third generation computer hardware systems are capable of supporting an IPAD implementation. Limitations are quantitative rather than qualitative. They include CPU speed, memory size, online mass storage capacity, etc. Fourth generation computers from CDC, Texas Instruments, Burroughs, and IBM will be much better suited to the expected computation and data transfer volume required in a fully utilized IPAD system. Additionally, the cost per operation (e.g., addition) will drop by a factor of four to six. Timing for 64 bit floating point add operations, for example, can be expected to fall below 20 nanoseconds, while the machine cost will remain roughly commensurate with today's CDC 7600 and IBM 370/195.

7.2.1 Host System

The characteristics of the hardware required to support an IPAD implementation were determined from the studies documented in volumes II and III. These studies provided an estimate of hardware capacity requirements in terms of:

- o CDC 6600 CPU hours
- o Input/output rate per CPU second (a measure of CPU - I/O dominance)
- o Storage required for program libraries
- o Storage required for data.

Program usage frequency was estimated in the studies from surveys of current and projected design practices for an organization similar to the Boeing Commercial Airplane Company and the Boeing Aerospace Company. These organizations include nearly 7500 design and production engineers, who were assumed to be involved in one detailed product design and seven concurrent preliminary design projects. The characteristics of these projects that affect hardware capacity are:

- o computer usage characteristics, including run frequency,
- o desired flow time, and
- o interactive terminal requirements.

Central Processing Unit - The CPU is the heart of a computer. It is generally the limiting factor with regard to solving extremely large problems, or allowing a great number of simultaneous timesharing users.

A number of computer manufacturers offer highly modular and interconnected computer systems with multiple instruction stream processors, multiple memory modules, and multiple input/output processors. This study considers the central processor to consist of an instruction processor, memory, and channels or memory ports. Computers with multiple instruction processors and memory modules are taken to contain one CPU with a memory and processing capacity determined by the total capacities of the modules. They should have the following characteristics:

a) Instruction Rate

Of all the simple parameters used to describe a computer system, the CPU instruction rate has the most effect on the speed with which the computer can produce results. Hence, any specification of instruction rate is tied directly to the volume of work expected in, for example, a 24 hour period. The IPAD host computer system capacity requirements, determined in Volume II, included the total number of CDC 6600 CPU hours required to support the Boeing Commercial Airplane and Boeing Aerospace Companies. The company mix is projected to consume 42.1 CDC 6600 hours, per 24 hour period. This is equivalent to an instruction rate of 10 to 14 million instructions per second (MIP), depending on central processor utilization.

b) Multiprogramming Capability

The IPAD system is inherently multi-simultaneous user oriented. This dictates the need for central memory write protection from concurrently running programs. Read and execute protection would be highly desirable but is not required. Also the hardware should support task switching in the order of microseconds.

c) Floating Point Precision

A CPU to support IPAD must be capable of at least 12 digit (40 bit) precision, with an exponent range of at least -40 to +40 (decimal). Some IPAD technical code modules will contain routines for solving very large systems of simultaneous equations, inverting very large matrices, solving nearly unstable differential equations, and other similar activities requiring high precision floating point arithmetic. In practice, the precision needed is dependent upon the problem formulation and the algorithm used. Research, design, and analysis applications on the CDC 6600 almost never require double precision (96 bits). However, double precision on the IBM 360 (53 bits) is often required and gives satisfactory results.

- d) **Input/Output Bandwidth**
Words input or output per CPU second on a CDC 6600 were obtained from the workload prediction study in Volume III. These predicted IPAD CPU I/O ratios are comparable to the average rates for the total work on Boeing's CDC 6600. These rates are:

- o 50000 words/CDC 6600 CPU second for IPAD application modules,
- o 56000 words/CDC 6600 CPU second for the Boeing 6600 job mix.

This is approximately 6 to 8 machine instructions executed for each character transferred. A typical IBM 370/165 installation for scientific work executes about 5 to 10 instructions per character transferred. An IPAD host computer should have an I/O bandwidth comparable to these figures.

- e) **Memory Size**
Application modules must be accommodated through an overlay technique, virtual memory, or a very large main memory. Virtual memory is preferred because it simplifies the design and operation of the data base manager. Application modules in the order of one million bytes on an IBM 360-370 or 320K octal words on a CDC 6600 are not unusual.

- f) **Character Representation**
The hardware representation of alphanumeric and special characters in the computer system is not important. It is important, however, that the computer system support a full upper and lower case alphabet and program constructable remote terminal control characters (e.g., line feed, backspace, etc.). ASCII-8 should be supported.

- g) **Upward and Downward Compatibility**
It is expected that the first IPAD system will be implemented on a medium to large scale computer. There will be a wide range of user performance demands at different installations. It would be an advantage to implement IPAD on one or more compatible families of computers. During the implementation period it would be desirable to have a dedicated member of the target family for software development and checkout.

Random Access Storage Devices - The IPAD system will utilize a spectrum of online data storage peripherals. The IPAD data management system is designed to take advantage of the speed and

capacity of these online data storage devices. High activity library entries, or fragments thereof, will be kept on high speed, low capacity devices. Inactive subtask libraries and currently active community library entries will be sorted on low speed, high capacity devices. Project and task histories, old experimental data, documents, and other such information will be retained on archival devices.

The specifications below are intended more as a definition of terms than a specific requirement. For fourth generation hardware, multiply the capacity-transfer rate product by 50.

a) High Speed, Low Capacity Storage

High speed, low capacity storage will be used primarily for program swapping, library indices, and small active job scratch data sets.

A high speed, low capacity device has a latency time of less than 10 milliseconds, a transfer rate of at least 1.5×10^6 8-bit bytes per second and a capacity of less than 10^7 8-bit bytes. Devices in this class include fixed head disks (IBM 2305), drums (UNIVAC FH-432), bulk core (CDC ECS), and future solid state memories using, for example, bulk MOS shift registers or magnetic domain techniques.

b) Low Speed, High Capacity Storage

Low speed, high capacity storage will be used for dictionaries, inactive subtask libraries, large portions of active subtask libraries, and currently active members of the community library. Library entries stored on this class of device are generally regarded as permanent, while the high speed, low capacity devices contain predominantly temporary copies.

A low speed, high capacity device has a latency time between 10 and 200 milliseconds, a transfer rate between 10^5 and 1.5×10^6 8-bit bytes per second, and a capacity between 10^7 and 10^{10} 8-bit bytes. Devices in this class include dismountable moving-arm disks (IBM 3330), non-dismountable moving-arm disks (CDC 6638), and magnetic strip storage (IBM 2321 data cell). In the near future some magnetic domain devices (Bubbles, DOT) and early holographic systems will be in this class.

c) Archival Storage

The use of archival storage is a distinguishing feature of the IPAD system. It will contain project

histories, experimental test data, backup versions of current library entries, and online data sets too large for other storage devices.

An archival storage device is defined as having a capacity greater than 10^{10} 8-bit types. Transfer rate should be at least 10^5 bytes per second. There are three marketed archival storage systems using different design approaches: Laser/aluminized mylar strip (Precision Instrument UNICON), large reel video tape (Ampex TBM), and cassette video tape (Grumman Masstape).

Unit Record Equipment - An IPAD host installation may have a complement of unit record devices:

- o Card readers (1 to 2000 cards per minute)
- o Card punches (3 to 600 cards per minute)
- o Line printers (\geq 1000 lines per minute).

The terminal orientation of IPAD greatly reduces the user's dependence on punch cards for program and data storage. For example, the computer programs used to verify and format the system design document in section 6.2. and Appendix A never existed on punch cards. The programs were entered, edited, and debugged entirely through alphanumeric CRT terminals. Card readers and punches will still be required in the future, but to a lesser extent than today.

Line printers, on the other hand, will remain important. One or more very high speed, single copy printers, would be satisfactory for program listings, checkout runs, and most mark-up and throw-away purposes. Also required is a high quality printer, similar to, but perhaps not as versatile as, today's page or photo composer used to compose books and newspapers. Such a device will be able to produce document quality tables and plots, if not entire documents.

Magnetic Tape Equipment - Half-inch magnetic tapes will be with us many years into the future. An IPAD system installation, while not dependent upon magnetic tape for its basic operation, will require the ability to accept data recorded by offline devices, non-IPAD computer systems, pre-IPAD computer programs, and IPAD users. The IPAD installation may also be called upon to create tapes for offline devices, very long term archival storage, and mailing to installations not reachable by a network. Since the primary purpose of tape on an IPAD system is communication, there must be both seven and nine track devices available.

Graphical Input/Output - Graphical input/output devices are not necessarily online to the central host computer. For example, a digitizer or programmable film reader for loading drawings into the system is generally a stand-alone device.

Two types of plotters are needed. The first is a high volume plotter to produce cheap, marginally accurate drawings (.01 inch), for check prints and inter-company communication. The other is a very accurate (.002 inch) drafting machine or flatbed plotter. Drawings produced would be used for manufacturing, mockup, wind tunnel, and other purposes.

In addition there may be a requirement for a microfilm plotter, which would be used for reducing and storing drawings on microfilm.

Reliability - The user community demands the freedom from considering the reliability of the IPAD host computer when planning projects. Computer designers have devised several techniques for detecting and correcting hardware errors, for example: parity bits, Hamming code correction, and automated voltage margin measurement and adjustment. They have served to improve the mean time between failure of the computer system hardware in the face of increasing logical complexity. The IPAD host computer must have a minimum mean time between failure of one to two weeks and should admit to prompt fault detection and repair.

Summary

CPU Instruction Rate	10 to 14 MIP for the total company mix. 3 to 4 MIP for Project I, subsonic commercial transport.
Multiprogramming	Memory write protect, read/execute protect desired.
Floating Point Precision	At least 12 digits with an exponent range of -40 to + 40 (decimal).
Input/Output Bandwidth	Transmit one character per 5 to 10 CPU instructions executed.
Memory Size	Allow 1 megabyte IBM 370, and 330K CDC 6600 programs to run.
Character Representation	Full upper and lower case alphabet. Terminal function control characters. Prefer 8 bit characters.

Upward and Downward Computability	Desirable.
Random Access Storage	Some of each. The quantity is to be determined at implementation.
o High Speed, Low Capacity	Transfer rate 1.5×10^3 KB. Capacity $\leq 10^7$ bytes.
o Low Speed, High Capacity	10^2 KB \leq transfer rate $\leq 1.5 \times 10^3$ KB. 10^7 bytes $<$ capacity $< 10^{10}$ bytes.
o Archival Storage	Transfer rate $\geq 10^2$ KB. Capacity $\geq 10^{10}$ bytes.
Unit Record Equipment	High speed card reader/punch. Very high speed line printers, low cost per page. High quality page or photo-composer for documents.
Magnetic Tape Equipment	7 and 9 track, industry compatible.
Graphical Input/Output	Digitizer or programmable film reader. High volume, marginally accurate paper plotters. High precision drafting machines. Possibly a microfilm plotter.
Reliability	Mean time between failure at least 1 to 2 weeks.

7.2.2 Terminals

Personal Terminals - The primary means of man-computer communication in the IPAD system is via terminals. This study has classified terminals into three main types:

- o Personal terminals like teletypes or teletype replacements,
- o Interactive graphics scopes, and
- o Remote job entry stations.

This study has defined a personal terminal to be a terminal operated by one person at a time primarily for two-way alphanumeric communication with the IPAD host computer. It will

be used for constructing and modifying CM's, OM's, jobs, and other library entries. Jobs run on the IPAD host system will, for the most part, be initiated, monitored, and interacted with using the personal terminal.

Most of the engineering interaction requirement is easily handled by purely alphanumeric devices like teletypewriters and CRT terminals. A CRT terminal should hold at least 20 lines of 72 characters. Any fewer almost demands a printer attachment. Silent printers and other terminal peripheral devices like cassette tape recorders, small x-y plotters, low-volume card readers, and simple digitizers have been identified as necessary or desirable in the engineering design process. Many manufacturers allow for switchable peripherals so that, for example, two or more alphanumeric CRT terminals may share a single printer. Personal terminals would be connected to the IPAD host via a dial-up telephone line. Whether the lines were public or private is an implementation decision. The use of dial-up telephone lines limits the bandwidth to approximately 2000 bits per second which is more than enough for information display but may just be adequate for a cassette tape attachment. Through experimentation this study has concluded 110 baud (ten character per second message transmission) is conducive to boredom, frustration, and work-arounds. Therefore, 300 baud is to be considered as a practical minimum line speed.

Interactive Graphics Terminals - The interactive graphics terminal differs from the personal terminal in its ability to display vector or line drawings. There is a definite requirement for interactive graphics in an IPAD system. Two classes of interactive graphics activity have been identified. One class is limited to simple keyboard input and is useful for displaying plots and graphs. The other class is the full interactive graphical input/output activity associated with geometric design and topological problems. Besides the display, hardware to support full interactive graphics includes a keyboard, lightpen, function keys, analog input devices like joysticks, graphical input devices like RAND tablets, and hardcopy attachments. Most interactive graphics terminals have self-contained minicomputers to refresh the display, poll the user input interfaces, communicate with the host system, and generally relieve the host system from minor interruptions.

Terminals containing minicomputers are generally able to interface with a large set of peripherals, including printers, small disks, slow half-inch tape drives, card readers and punches, and telecommunications gear.

Remote Job Entry Terminals - In a geographically distributed engineering community there are flowtime problems associated with bulk manual and vehicular transport of computer input and

output. A remote intelligent terminal connected to the IPAD host system via a wideband line would provide medium speed printers, punches, card readers, and possibly tapes within walking distance of users. Large volumes of newly generated data will, for the near future, continue to be in the form of punched cards. The remote job entry terminal allows the remote user to enter his data into the IPAD data base and obtain a printed copy for visual checking and backup. Its medium speed printers complement the personal terminal printers when large data sets are to be printed. The remote job entry terminal may also act as a message concentrator to minimize line costs for local personal terminals.

7.2.3 Networks

Networks of interconnected computer systems will be prevalent in the future as users recognize their advantages. Computing power will be distributed, much like the electric power industry where failure of one component or subsystem may be bypassed with a negligible interruption of service. Access and response time will more closely approach optimum when work can be partitioned among the network's computers.

Through network facilities, specialized installations such as ILLIAC IV at NASA Ames, will become available to remote users.

The greatest benefit computer networks hold for IPAD is inter-installation communication. Government agencies may pass specifications to contractors and receive reports. Contractors and sub-contractors can share computer programs and data, guarantee consistency of configuration, stress levels, etc. Public libraries of computer programs, standard reference data like atmospheric properties, and cross indexes of technical literature will serve to organize the engineering design process to an unprecedented degree.

For the near future, an IPAD host system could be connected into a wideband packet switching, store and forward network similar to the ARPA net. ARPA uses 50 kilobit dedicated lines between nodes and special-purpose minicomputers to interface the local computer system to the network. The minicomputer is responsible for all data transmission and error management. In addition it can reconfigure the network in the event of a hard line failure.

During the implementation of the first few IPAD systems, careful thought will have to be given to the expected growth of networks and network traffic. Flexibility of the network's plan must be sufficient to allow individual hosts to evolve separately.

7.3 IPAD HOST COMPUTING SYSTEM USING A CDC 6600 (CYBER 74)

The host configuration in figure 7.1 is recommended for an IPAD implementation in a CDC 6600 installation. The configuration is based upon the requirements from Volume III and those given in section 7.2. The recommended operating system is KRONOS 2.1, primarily on the basis of its orientation to terminal operations. SCOPE 3.4 would also be acceptable. Two critical features, in terms of implementation schedule, absent in both of these systems are:

- a) multitasking within a single users terminal control and
- b) the ability to log-off the terminal with the job active for later log-on and reconnection.

Implementation of these features would currently involve operating system modifications.

Multitasking is an operating system feature which allows a running program to command the operating system to execute another program, usually in parallel with the originating program. The originating program may obtain status information through the operating system and may abort the subordinate program and continue in execution itself. This process of "attaching" other programs may be nested, or treed, to many levels.

Specific equipment for personal terminals is not included because of the rapidly changing technology. However, the characteristics to support a selection at implementation time are listed. The interactive graphics equipment is an example, rather than a hard requirement because the specific needs are unknown at this time.

The equipment list for this host configuration is given below. A schematic diagram is given in figure 7.1.

- a) CPU and system control equipment
 - o CYBER 74-18 CPU (131,072 CM)
 - o 6612 system CRT console
 - o PP option for 14 PPU's and 18 data channels
 - o Three each 844-2 disc drives with three each 7054 controllers (to be used for system storage and job swapping)
- b) Online data storage for the user
 - o Ten each 844-2 disc drives with three each 7054 controllers
 - o Five each 821-2 disc drives with two each 3553 controllers and two each 6681 data channel converters

- c) Tape drives for system and users
 - o Two each 657-4 tape drives (7-track)
 - o Four each 659-4 tape drives (9-track)
 - o Two each 6681 data channel converters
 - o One each 3528 controller
- d) Graphics
 - o One each 1700 auxiliary computer with a 6674 controller
 - o Two each 274 display CRT's each with a 1744 controller
 - o One each tape drive with a controller
 - o plotters (off line, CRT slave, connected to remote batch) drafting machines (offline)
- e) Remote batch entry
 - o Two each 732 terminals (4800 baud) with 1 card reader and printer each
 - o Two each 732 terminals (9600 baud) with 1 card reader and printer each
 - o One each 6671 controller
- f) Personal terminals
 - o Two each 6676 controllers
 - o 100 personal terminals
 - . CRT
 - . 600/1200 baud
 - . minimal vector capability
 - . TTY interface compatibility
 - o 50 shared printers for the terminals
 - o 25 shared cassette drives for the terminals
- g) Local batch
 - o One each 405 card reader with 3445 controller
 - o One each 415 card punch with 3446 controller
 - o Two 512 printers with 3455 controllers
 - o One 6681 data channel converter

The CDC configuration of figure 7.1 will support the computational load from two design projects similar to Project 1 (subsonic transport in Volume II). However, the peak efforts, during level 4, in Project 1 (configuration refinement) would have to be staggered about three months to prevent saturation. Project 2 (supersonic transport) would require nearly one and one third greater capacity than this host configuration for either level 3 (configuration sizing) or level 4 (configuration refinement).

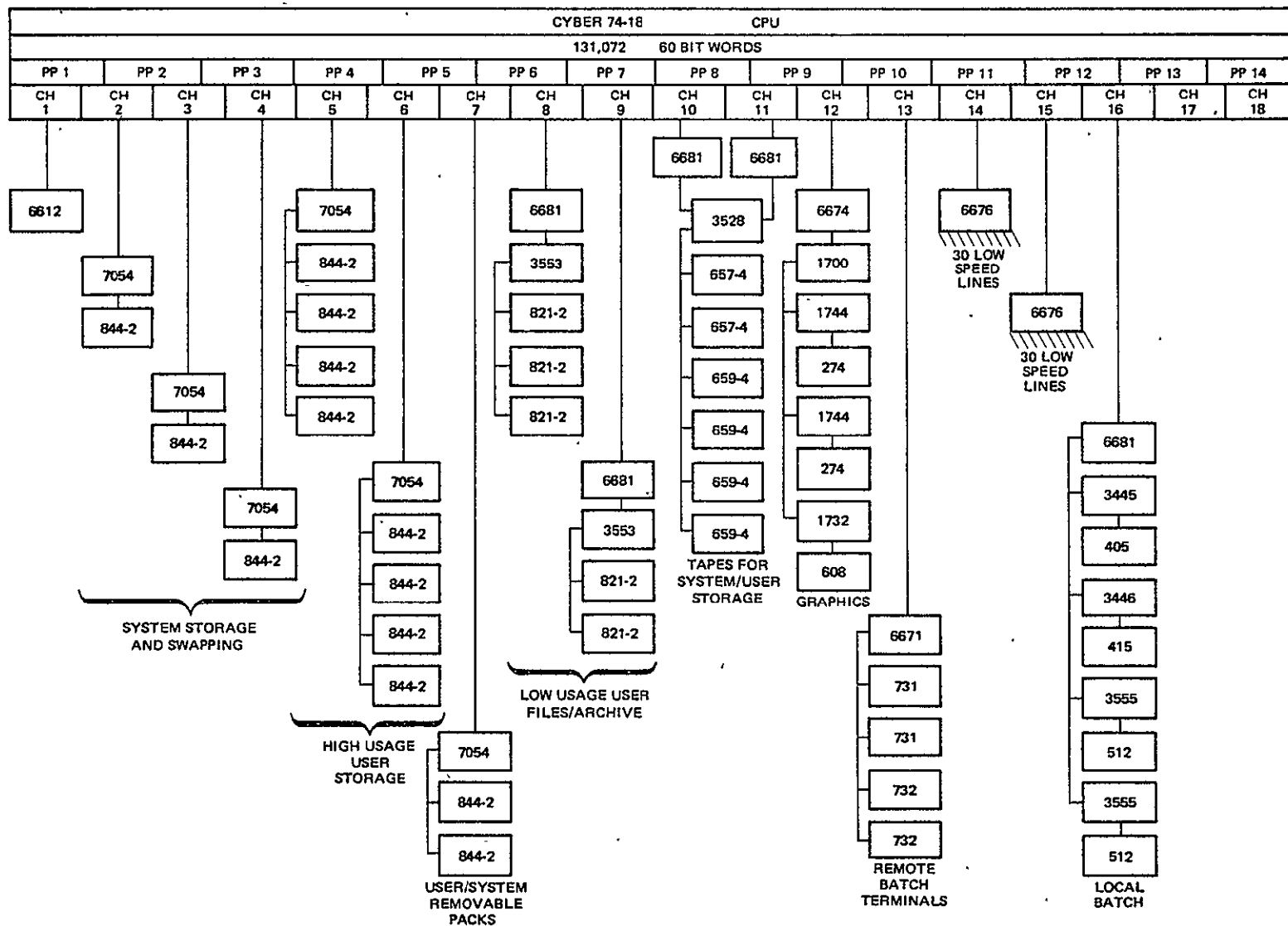


Figure 7.1 IPAD Host System—CDC 6600 (CYBER 74)

ORIGINAL PAGE IS
OF POOR QUALITY

7.4 IPAD HOST COMPUTING SYSTEM USING AN IBM 370/168

The host configuration in figure 7.2 is recommended for an IPAD implementation in an IBM 370/168 installation. This configuration is based on the IPAD host capacity requirements from Volume III and the host hardware requirements from section 7.2.

Due to the essential time sharing nature of the IPAD system, OS/VS2 with TSO (Time Sharing Option) is recommended. VS2 was selected because TSO on VS2 allows 42 simultaneous active regions, while TSO on MVT allows only 14. The two critical features, in terms of implementation time, of the IPAD design which currently will require system (TSO) modifications are:

- a) implementation of "PAUSE" and
- b) the ability to log-off the terminal with the job active for later log-on and reconnection.

The "PAUSE" command in the IPAD system enables a terminal user, at any time, to interrupt an executing program. He may then give a "GO" command to resume execution, or he may enter some other IPAD command. At the completion of the inserted IPAD command he may resume execution of the interrupted program. This process is nested and a user may have several programs in suspension at one time.

The technology of personal terminals is changing so rapidly that a specific selection at this time would serve no purpose. Particular terminals will be selected at implementation time. Four 2250 graphics consoles have been included. This number is variable depending on the level of sophistication of the graphics technology at an individual installation.

The equipment list for this configuration is given below. A schematic diagram is given in figure 7.2

- a) CPU and Miscellaneous System Equipment
 - o 3168KJ CPU with
 - . High speed multiply
 - . Buffer expansion
 - . 3 million bytes
 - o 3067 power supply
 - o 3066 CRT operator's console
- b) Channels
 - o One 2880-1 Block multiplexer channel
 - o Two 2880-2 Block multiplexer channels

- o One 2860-2 Selector channel
 - o One 2870 Multiplexer channel with one selector subchannel
- c) Online Data Storage for the User and System
- o One 2305 Fixed head disk with its 2835 controller
 - o Three 3333/3330 8 spindle disk systems with three 3830-2 controllers
 - o One 3333/3330 16 spindle disk system with 3830-2
- d) Tape Drives for the User and System
- o Two 3420-5 7-track, multi-density with 3803 controller
 - o Four 3420-5 9-track, 800/1600EPI with 3830 controller
- e) Graphics
- o Four 2250-3 Interactive graphics terminals
- f) Remote Batch Entry
- o One 2702 Transmission unit, 2-4 high speed lines
- g) Personal Terminals
- o One 2703 Transmission unit, 60 low speed lines
 - o 100 personal terminals
 - . CRT
 - . 600 baud
 - . Minimal vector capability
 - . TTY compatible
 - o 50 shared terminal printers
 - o 25 shared terminal cassette units
- h) Local Batch
- o One 2821-5 Unit record controller
 - o One 2540 Card reader/punch
 - o Two 1403-N1 Printers, with Universal Character Set feature

An IPAD system running on this IBM host configuration would be capable of supporting two to three Project 1 (Subsonic Transport) efforts in parallel. As with the CDC 6600 host configuration, it would be very important to schedule the load peaks of the individual projects to minimize saturation. This configuration could handle levels 2 and 3 of Project 2 (Supersonic Transport) provided some rescheduling and stretchout were done to reduce the peak capacity requirements given in Volume III.

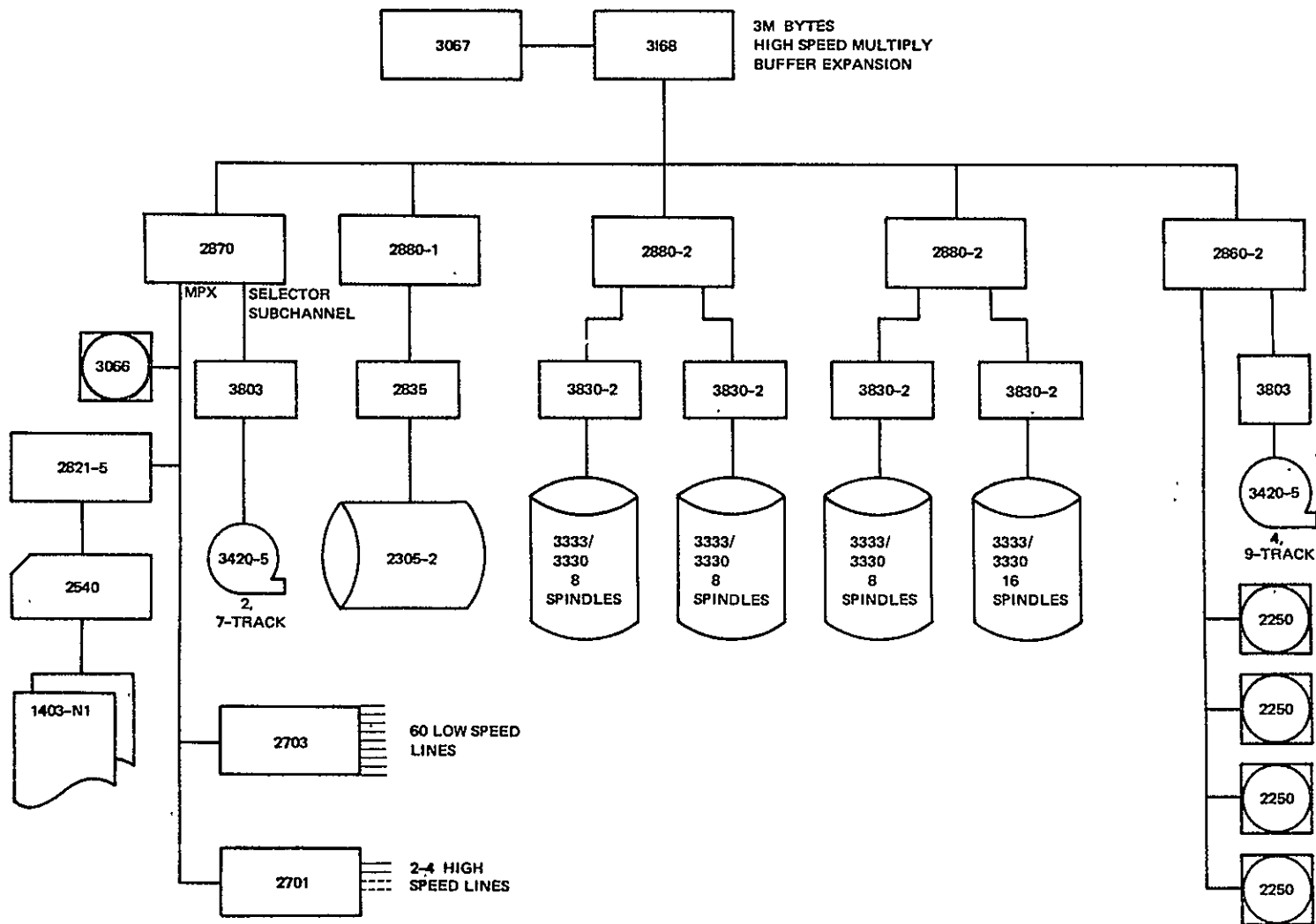


Figure 7.2 IPAD Host System - IBM 370/168

REFERENCES

- (1) Mills, H. D., "Mathematical Foundation for Structured Programming", IBM Report FSC72-6012, Feb. 1972
- (2) IBM Report FSC71-5108, "Chief Programmer Teams: Principles and Procedures", June, 1971
- (3) E. Glaser, et al., Proceedings of 1971 COMPCON Conference, Sept. 1972, Set of Articles on LOGOS System, Case-Western Reserve University
- (4) Association for Computing Machinery, CODASYL Data Base Task Group, April 1971 Report.
- (5) CODASYL Stored Data Definition and Translation Task Group, "An Approach to Stored Data Definition and Translation", Sept., 1971, and unpublished paper by Taylor, R. W. "The Translation Process", U. of Massachusetts, Sept., 1972.
- (6) Miller, Robert B., "Response time in Man-Computer Conversational Transactions", AFIPS Conference Proceedings for Fall Joint Computer Conference, 1968
- (7) Martin, James Systems Analysis for Data Transmission, Englewood Cliffs, New Jersey, Prentice-Hall, 1972, pp 61-122

APPENDIX A

DETAILED SYSTEM DESIGN SPECIFICATIONS

Structured programming is a formal work dealing with software engineering and hardware-software system design and development (ref. 1, 2, and 3). The objective of this work is to transform the development of computer systems from a seat-of-the-pants art to a disciplined technology. This approach has been utilized to develop the IPAD system design.

The structured programming approach is a top down design method in which the design proceeds from the general to the specific. Each refinement is a level in the system design. Tree structure diagrams give the system functional components in levels of increasing detail. The nodes at any one level in the tree structure are states of activity for the system. The entire system is included in the total set of nodes of each level, and in fact, higher level nodes are summaries of lower level nodes.

Transition diagrams describe how the system components, at each level, are functionally related. The diagrams also specify the conditions under which there will be a transition or state change within a node or from one node in the tree to another node at the same level. These transition conditions are (1) the input data or conditions that trigger the transition and (2) the output data or results existent in the system at the time the transition is made. Figure A.1 is a sample tree structure and transition diagram for a three level system.

The IPAD system design given in appendix A follows the general form described above. In level 1, twenty-nine nodes or states are described. Except for a few level 1 states dealing with hardware or host operating system protocol, the level 1 states are each refined into level 2 states. The level 2 states are, in turn, broken out into level 3 states, and so on. The emphasis in the design was placed upon consistency in detail rather than consistency in levels documented. Hence, there are differences in the depth or number of levels reached in some of the tree branches.

While the design as presented is in top down form, the actual design process does not proceed monotonically. Generally, design at level n will result in a review of some elements of the design at level $n-1$, $n-2$, etc. The advantage of the method is that the examination of effects is an orderly process and the consequences of the iterative design process are highly visible.

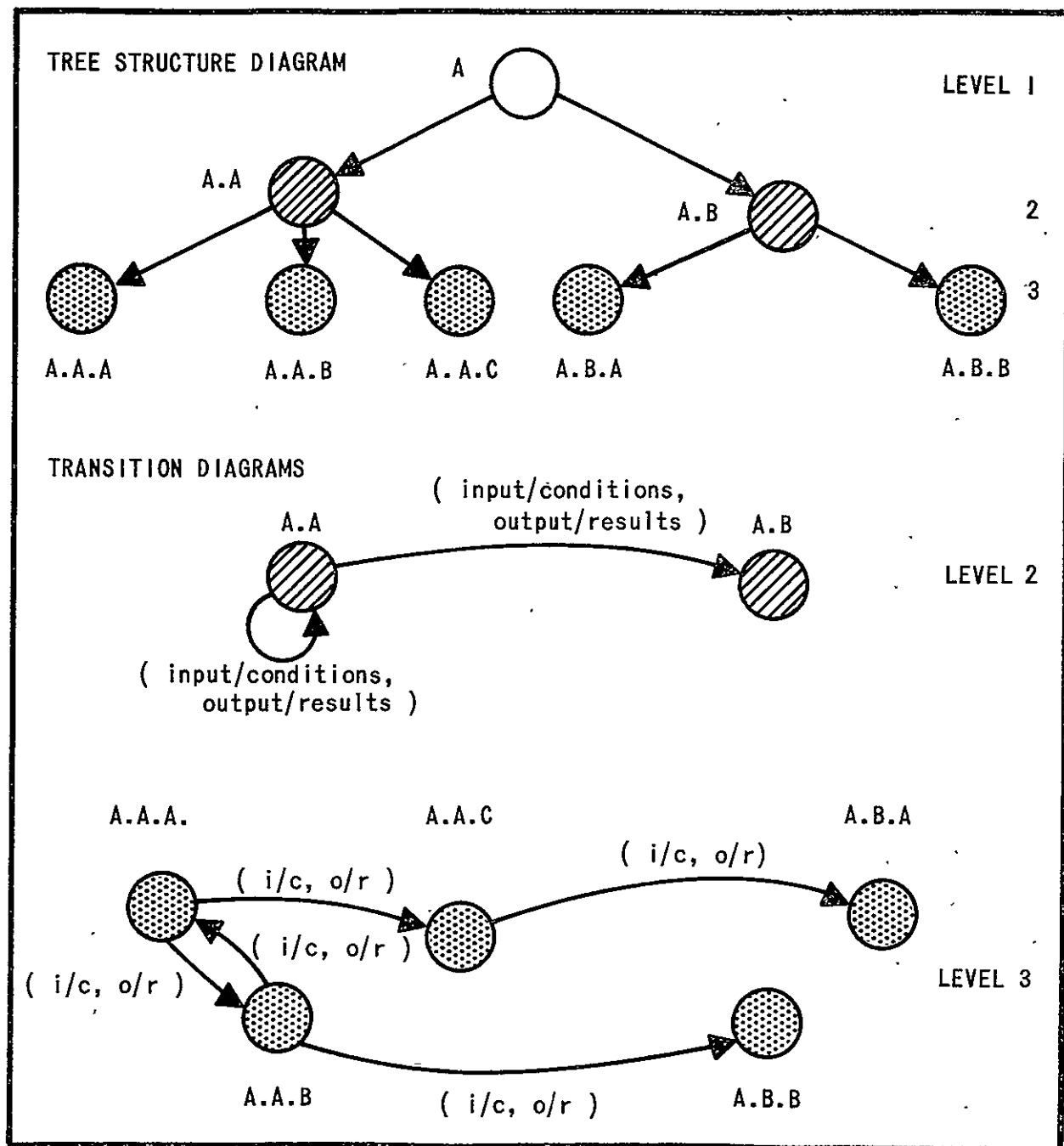


Figure A.1 Structured Programming Diagrams

Each level in appendix A contains the following diagrams and tables:

State Description Tables - Three pieces of information are given for each node:

- a) Short Structured Name - This name consists of a set of one or two alphabetic characters catenated in the form

rs
rs.tu
rs.tu.vw
etc.

The syllable position denotes a level. For example, if node A is at level 1 then node A.B would be at level 2 and would be a state of node A. Hence, the tree diagram can be formed from the short structured names. There is no requirement that these names be in sequence, i.e., the existence of node A.B and node A.D does not presuppose the existence of node A.C.

- b) Long Name - This name is descriptive of the function of the node. For example, node E has the long name "Subtask Set-Up."
- c) Description - Several sentences, in summary form, describing the capabilities of the node.

Allowed Transition Tables - This is a tabular representation of the connections between the nodes having a common parent at the next higher level. The states from which and to which transitions are made, along with the corresponding references to the input/condition and output/result tables which follow, are given. A bent arrow is used to flag entry and exit points from the parent node. When exits are shown, the level of the state exited to may be at a higher level than the state being exited from depending upon the level of tree structuring completed.

Transition Diagrams - These are a graphical representation of the allowed transition tables. They can be constructed from the transition tables and are valuable for visualizing relationships.

Input/Conditions List Tables - This is a list of the input or conditions that trigger a transition or change of state. This list should be used in conjunction with the Allowed Transition tables.

Output/Result List Tables - This is a list of the output or results that are existent in the state when a transition is made. This list should also be used in conjunction with the Allowed Transition tables.

Abbreviations are not used in level 1. They are used in lower levels to facilitate writing. Definitions of abbreviations are given in section 4.0 of Volume IV. The text part of appendix A was created by a computer program from data supplied by the system designers. This computer program checked for consistency to ensure that transitions were made between valid states and that lower level states were correctly referenced to higher level states.

Tree diagrams are not included. They can be constructed from the structured names.

Figures A.1 and A.2 are the level 1 transition diagram. Node F is repeated on figure A.3 for reference. These figures should be read in conjunction with the State Descriptions, Allowed Transitions, Input/Condition List and Output/Result List following the diagrams. Transition diagrams for lower levels are included with the level.

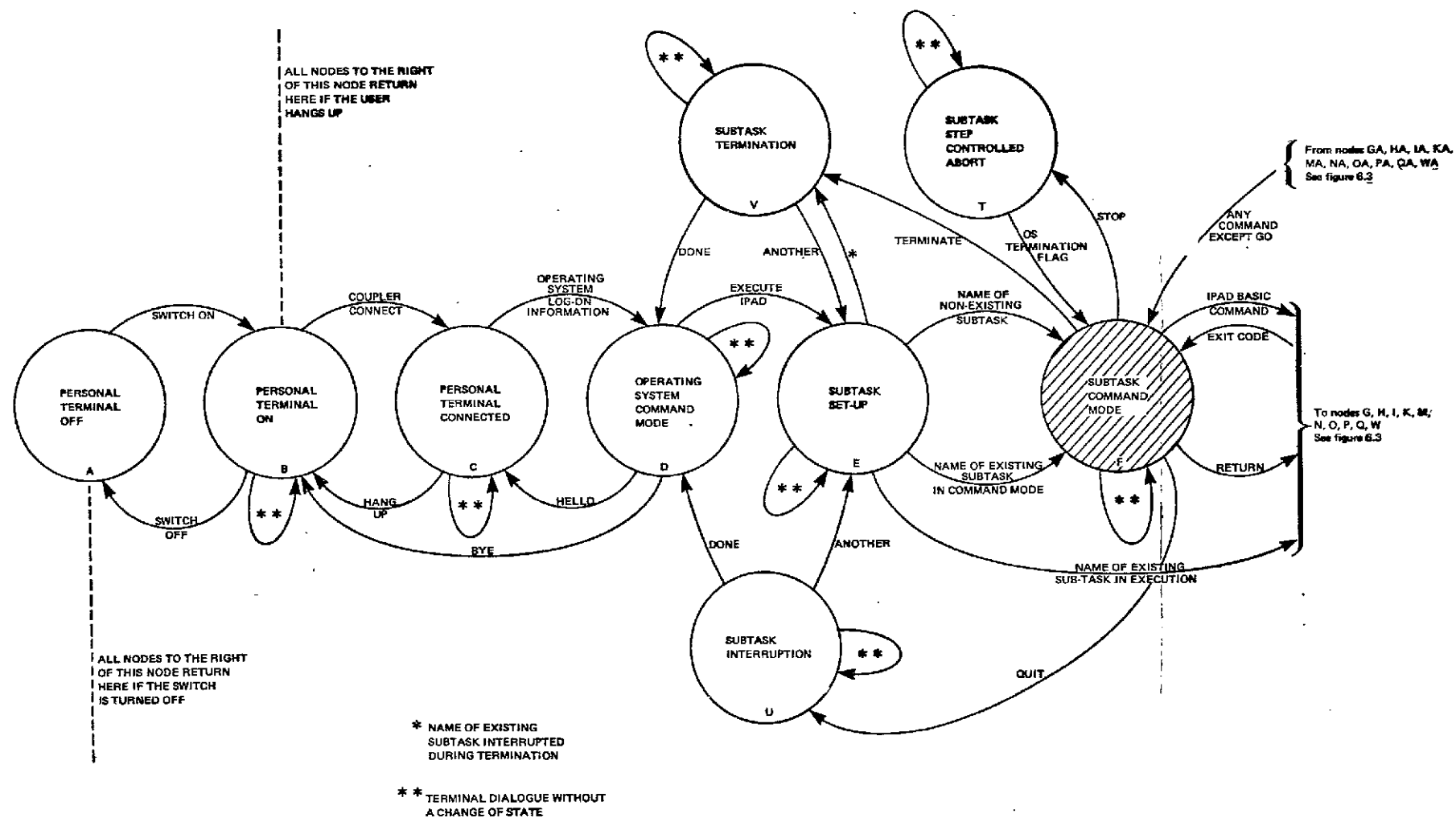


Figure A.2 IPAD System Design Level 1 Transition Diagram

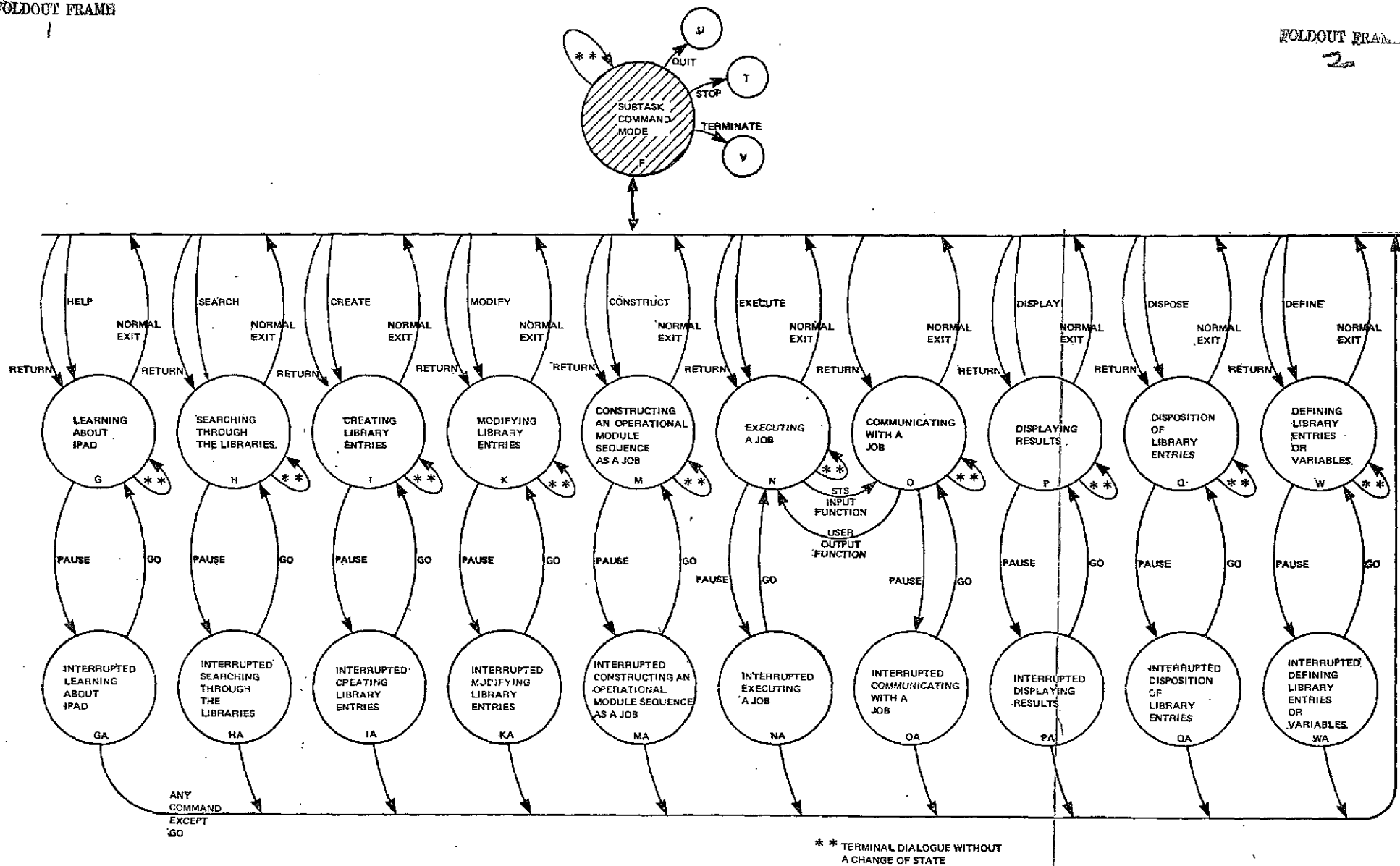


Figure A.3 IPAD System Design Level 1
Transition Diagram (Cont'd)

COMPONENTS OF IPAD LEVEL ONE

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

A PERSONAL TERMINAL OFF

THE EQUIPMENT IS NOT ACTIVE.

B PERSONAL TERMINAL ON

THE EQUIPMENT IS ACTIVE BUT NO DATA PATH TO THE
COMPUTER EXISTS. THE EQUIPMENT IS A PERSONAL TERMINAL,
NOT A REMOTE JOB ENTRY TERMINAL, BUT MAY BE AUGMENTED
WITH PERIPHERAL DEVICES SUCH AS CASSETTE TAPE, PRINTER,
PLOTTER.

C PERSONAL TERMINAL CONNECTED

THERE NOW EXISTS A TWO-WAY DATA PATH BETWEEN THE
TERMINAL AND THE COMPUTER

D OPERATING SYSTEM COMMAND MODE

THE USER IS NOW ABLE TO ENTER COMMANDS TO THE TIME
SHARING SYSTEM IN THE HOST OPERATING SYSTEM

F SUBTASK SET-UP

THE USER IS NOW IN COMMUNICATION WITH IPAD AND HE
IS EITHER INITIATING A NEW SUBTASK OR CONTINUING AN
OLD ONE. IN EITHER CASE, THE NET RESULT WILL BE THE
ESTABLISHMENT OF HIS ACTIVE SUBTASK LIBRARY.

F. SUBTASK COMMAND MODE

THE USER IS NOW ABLE TO ISSUE IPAD BASIC COMMANDS
TO ADVANCE HIS SUBTASK WORK.

IPAD: LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE LONG NAME AND TEXT

G. LEARNING ABOUT IPAD

THE ACTIVITY OF GAINING INFORMATION ABOUT IPAD
EITHER AS A TAUGHT COURSE OR AS HELP WITH A SINGLE
COMMAND OR MODULE.

H SEARCHING THROUGH THE LIBRARIES

THE PROCESS OF SCANNING DICTIONARIES AND DIRECT-
ORIES TO IDENTIFY AND LOCATE INFORMATION IN THE IPAD
DATA BASE.

I CREATING LIBRARY ENTRIES

THE PROCESS OF INSERTING DATA (NUMERICAL AND CH-
ER) INTO THE IPAD DATA BASE RESULTING IN NEW LIBRARY
ENTRIES(LE). INCLUDED IS THE ENTERING OF SOURCE CODE
FOR CODING MODULES(CM), INFORMATION FOR STORED DATA DEF-
INITIONS(SDD), INSTANCES OF DATA SETS(DS), DISPLAY MENUS
(DM), AND THE INSTANCE OF THE SYSTEM DATA SET CONTAIN-
ING ACCESS AND PERMISSION CODES.

K MODIFYING LIBRARY ENTRIES

ALTERING CURRENTLY RESIDENT LIBRARY ENTRIES. THIS
CAN INVOLVE CHANGES TO ANY VALID IPAD LIBRARY ENTRY
TYPE.

M CONSTRUCTING A JOB

ARRANGING AVAILABLE CODING MODULES(CM) INTO OPER-
ATIONAL MODULES(OM), OPERATIONAL MODULES INTO JOBS, AND
OPERATIONAL MODULES AND PREVIOUSLY DEFINED JOBS INTO
NEW JOBS.

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
N	EXECUTING A JOB ACTIVATING A PREVIOUSLY CONSTRUCTED JOB
O	COMMUNICATING WITH A JOB BEING INTERACTIVE WITH A USER CONSTRUCTED JOB
P	DISPLAYING RESULTS SCANNING, CHECKING, AND INTERROGATING INFORMATION CONTAINED IN LIBRARY ENTRIES OF ANY TYPE.
Q	DISPOSITION OF LIBRARY ENTRIES TRANSFERRING LIBRARY ENTRIES BETWEEN IPAD LIB- RARIES, SENDING ITEMS OUTSIDE OF IPAD(OFFLINE, OR VIA A COMMUNICATION NETWORK), AND REMOVAL OF UNWANTED LIBRARY ENTRIES FROM THE DATA BASE.
T	SUBTASK STEP CONTROLLED ABORT THE TERMINATION OF THE CURRENTLY INTERRUPTED SUB- TASK STEP.
U	SUBTASK INTERRUPTION ACTION AIMED AT TEMPORARY INTERRUPTION OF THE SUB- TASK ACTIVITIES WITH THE INTENT OF RE-STARTING AT A LATER TIME AT THE PRECISE POINT OF INTERRUPTION.

IPAD LEVEL ONE
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
-------	--------------------

V	SUBTASK TERMINATION
---	---------------------

THE USER HAS COMPLETED THE DEFINED SUBTASK AND NOW DESIRES TO DISPOSE OF ALL REMAINING INFORMATION, LOG THE TERMINATION IN THE PROJECT PLANS, AND ISSUE ANY REQUIRED REPORTS.

W	DEFINING LIBRARY ENTRIES OR VARIABLES
---	---------------------------------------

A DEFINITION IS A DICTIONARY ENTRY WHICH CONTAINS THE MEANING OF A VARIABLE OR A LIBRARY ENTRY AND CROSS REFERENCING INFORMATION. ALL COMMUNITY LIBRARY ENTRIES AND VARIABLES REFERENCED IN DATA SETS REQUIRE DEFINITIONS. DICTIONARY ENTRIES ARE OPTIONAL FOR SUBTASK LIBRARY ENTRIES.

GA	INTERRUPTED LEARNING ABOUT IPAD
----	---------------------------------

THIS IS THE STATE IMMEDIATELY FOLLOWING A PAUSE DURING LEARNING ABOUT IPAD. EACH OF THE STATES G, H, I, K, M, N, O, P, Q, AND W HAVE A SIMILARLY ASSOCIATED STATE.

HA	INTERRUPTED SEARCHING THROUGH LIBRARIES
----	---

IA	INTERRUPTED CREATING LIBRARY ENTRIES
----	--------------------------------------

KA	INTERRUPTED MODIFYING LIBRARY ENTRIES
----	---------------------------------------

MA	INTERRUPTED CONSTRUCTING A JOB
----	--------------------------------

NA	INTERRUPTED EXECUTING A JOB
----	-----------------------------

IPAD LEVEL ONE
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
QA	INTERRUPTED COMMUNICATING WITH A JOB
PA	INTERRUPTED DISPLAYING RESULTS
QA	INTERRUPTED DISPOSITION OF LIBRARY ENTR.
WA	INTERRUPTED DEFINING LIBRARY ENTRY/VAR

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P A	B	1	1
B	A	14	14
	C	2	2
C	A	14	13
	B	13	13
	D	3	3
D	A	14	13
	B	13	13
	B	15	15
	C	12	12
	E	4	4
E	A	14	20
	B	13	16
	F	5	5
	F	6	6
	V	16	6
F	A	14	17
	B	13	17
	G	17	22
	G	34	39
	H	13	22
	H	34	39
	I	19	22
	I	34	39
	K	21	22
	K	34	39
	M	23	22
	M	34	39
	N	24	22
	N	34	39
	O	34	39
	P	27	22
	P	34	39
	Q	28	22
	Q	34	39
	T	9	9
	U	8	8
	V	7	7
	W	20	22
	W	34	39

ORIGINAL PAGE IS
OF POOR QUALITY.

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
G	A	14	41
	B	13	40
	F	35	42
	GA	31	36
H	A	14	41
	B	13	40
	F	35	42
	HA	31	36
I	A	14	41
	B	13	40
	F	35	42
	IA	31	36
K	A	14	41
	B	13	40
	F	35	42
	KA	31	36
M	A	14	41
	B	13	40
	F	35	42
	MA	31	36
N	A	14	41
	B	13	40
	F	35	42
	O	25	22
	NA	31	36
O	A	14	41
	B	13	40
	F	35	42
	N	26	22
P	OA	31	36
	A	14	40
	B	13	40
	F	35	42
Q	PA	31	36
	A	14	41
	B	13	40
	F	35	42
T	QA	31	36
	A	14	18
	B	13	18

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
U	A	14	19
	B	13	19
	D	11	11
	E	10	10
V	A	14	21
	B	13	21
	D	11	11
	E	10	10
W	A	14	40
	B	13	40
	F	35	42
	WA	31	36
GA	A	14	40
	B	13	41
	F	33	38
	G	32	37
HA	A	14	40
	B	13	41
	F	33	38
	H	32	37
IA	A	14	40
	D	13	41
	F	33	38
	I	32	37
KA	A	14	40
	B	13	41
	F	33	38
	K	32	37
MA	A	14	40
	B	13	41
	F	33	38
	M	32	37
NA	A	14	40
	B	13	41
	F	33	38
	N	32	37
OA	A	14	40
	B	13	41
	F	33	38
	O	32	37

IPAD LEVEL ONE
(CONTINUED)

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
PA	A	14	40
	B	13	41
	F	33	38
	P	32	37
QA	A	14	40
	B	13	41
	F	33	38
	Q	32	37
WA	A	14	40
	B	13	41
	F	33	38
	W	32	37

IPAD LEVEL ONE
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SWITCH TURNED ON
2	DIAL UP
3	VALID OS LOG ON INFORMATION IN THE PROPER SEQUENCE
4	VALID OS COMMAND TO EXECUTE IPAD
5	VALID SUBTASK IDENTIFIER FOR A NON EXISTING SUBTASK
6	VALID SUBTASK IDENTIFIER FOR AN EXISTING SUBTASK
7	TERMINATE
8	QUIT
9	STOP
10	ANOTHER
11	DONE
12	HELLO
13	USER HANGS UP
14	SWITCH TURNED OFF
15	BYE
16	SUBTASK RECORDS SHOWING AN INTERRUPT OCCURRED DURING THE SUB TASK TERMINATION
17	HELP
18	SEARCH
19	CREATE
20	DEFINE
21	MODIFY
22	CONSTRUCT
23	EXECUTE
24	CONDITION CODE SHOWING TERMINAL INPUT IS REQUIRED
25	LAST LINE OF USER INPUT
26	DISPLAY
27	DISPOSE
28	PAUSE
29	GO
30	ANY COMMAND EXCEPT A GO
31	RETURN
32	EXECUTION COMPLETED - NORMAL EXIT

IPAD LEVEL ONE
(CONTINUED)

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	PHONE LINE CONNECT
3	VALID OPERATING SYSTEM LOG-ON INFORMATION
4	OPERATING SYSTEM COMMAND TO EXECUTE IPAD LOG-ON PROGRAM
5	ESTABLISHMENT OF A NEW SUBTASK LIBRARY IN THE LL
6	THE OLD SUBTASK LIBRARY IN ACTIVE FORM
7	OS COMMAND TO EXECUTE THE SUBTASK TERMINATION PROGRAM
8	OS COMMAND TO EXECUTE THE SUBTASK INTERRUPTION PROGRAM
9	OS COMMAND TO EXECUTE THE SUBTASK STEP INTERRUPTION PROGRAM
10	SUBTASK INTERRUPTION COMPLETE
11	-
12	VALID LOG OFF INFORMATION
13	PHONE LINE DISCONNECT
14	-
15	-
16	SUBTASK LIBRARY ENTRY RESTORED TO ORIGINAL STATE
17	A PROCEDURE WILL BE EXECUTED EQUIVALENT TO THE FOLLOWING INPUTS - 3,13.
18	COMPLETION OF TERMINATION, THEN PROCEEDING PER OUTPUT 17
19	COMPLETION OF INTERRUPTION, THEN PROCEEDING AS IF INPUT 13 HAD BEEN RECEIVED.
20	OUTPUTS 16 AND 13
21	HOLDING OF THE TERMINATION INTACT SO IT WILL BE ENTERED UPON RETURN, THEN PROCEEDING AS IF INPUT 13 HAD BEEN ENCOUNTERED
22	PARSED COMMAND, UPDATED ACTIVITY RECORD
30	EXPLANATORY TERMINAL OUTPUT(IF NEEDED), INPUT REQUEST
31	INPUT TO SUBTASK STEP
36	CURRENT SUBTASK STEP INTERRUPTED IN RE-STARTABLE FORM
37	-
38	POINTER TO INTERRUPTED SUBTASK STEP PLUS RESTART INFORMATION
39	OS COMMAND TO RE-START FROM PRECEEDING PAUSE
40	IPAD PROCEDURE EXECUTED CONSISTING OF A PAUSE, QUIT, DONE,BYE
41	IPAD PROCEDURE EXECUTED CONSISTING OF QUIT,DONE,BYE
42	NORMAL EXIT CODE

LEVEL 2
COMPONENTS OF STATE E
SUBTASK SET-UP

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

E.A 1PAU LOG-ON

USER SUPPLIES HIS USER ID, PASSWORD, AND
SUBTASK IDENTIFIER. THE SYSTEM WILL CHECK FOR
USER VALIDITY AND THE EXISTENCE OF THE SUBTASK.

E.B RE-ACTIVATE OLD SUBTASK

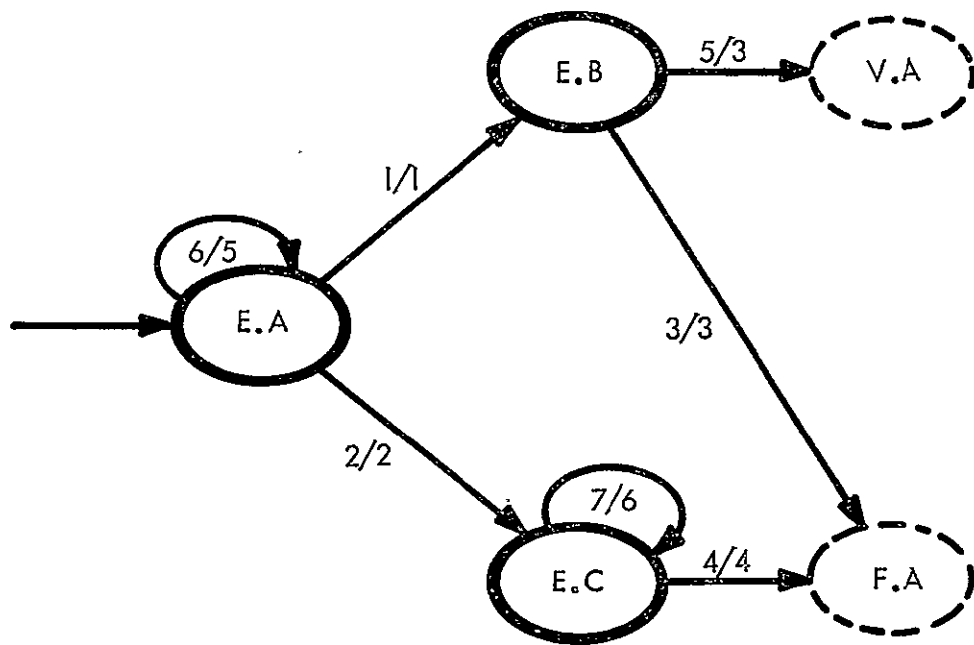
GIVEN AN INACTIVE SUBTASK NAME, RESTORE THE
SUBTASK LIBRARY TO ITS PRE-INTERRUPTED STATE. SPECIAL
CONSIDERATIONS ARE NECESSARY IF THE INTERRUPT OCCURRED
DURING SUBTASK TERMINATION.

E.C CREATE NEW SUBTASK

INITIATE A SUBTASK WILL GENERALLY DEPEND UPON THE
EXISTENCE OF PROJECT PLANS REFERENCING SUCH A SUBTASK.
ADMINISTRATIVE CONTROL WILL BE EXERCISED PRIMARILY
THROUGH THIS MECHANISM. SUBTASKS WITH NO FORMAL
PROJECT RELATIONSHIP MAY BE HANDLED THROUGH A SPECIAL
CATCH-ALL PROJECT.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P E.A	E.A	6	5
	E.B	1	1
	E.C	2	2
E.B	P F.A	3	3
	P V.A	5	3
E.C	E.C	7	6
	P F.A	4	4



LEVEL 2 TRANSITION DIAGRAM

STATE E : SUBTASK SET UP

E : SUBTASK SET-UP
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	USER ID AND THE NAME OF A CURRENTLY INACTIVE SUBTASK
2	USER ID AND A SUBTASK NAME NOT RESIDING IN THE CL
3	VALID SUBTASK LE IN THE CL
4	ALL INFORMATION NECESSARY TO INITIATE A SUBTASK.
5	INDICATION IN THE SUBTASK LE THAT TERMINATION WAS IN PROGRESS WHEN THE USER HUNG UP OR TURNED THE SWITCH OFF.
6	INSUFFICIENT VALIDITY CHECK INFORMATION
7	INITIALIZATION INFORMATION FOR USER OPTIONS AND/OR PROJECT REQUIREMENTS

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	LOCATION OF THE SUBTASK TO BE ACTIVATED
2	NAME OF THE NEW SUBTASK AND POINTER TO PROJECT PLANS
3	SUBTASK LIBRARY RESTORED TO INTERRUPTION TIME STATUS. ONE EXCEPTION IS IF A JOB WAS LEFT IN EXECUTION AT THE TIME OF INTERRUPTION AND IS NOW INACTIVE. IN SUCH A CASE, THE NEW STATUS WILL ACCOUNT FOR THE INTERIM ACTIVITY.
4	SUBTASK LIBRARY INITIALIZED CONSISTENTLY WITH THE PROJECT PLANS.
5	EXPLANATION OF ADDITIONAL CHECK INFORMATION REQUIRED
6	ADDITIONS TO THE STL SET UP

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
E.A	U.B
	U.C
	V.C

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 2
COMPONENTS OF STATE F
SUBTASK COMMAND MODE

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

F.A REQUEST USER INPUT AND INTERPRET COMMAND

THE SYSTEM WILL PROMPT THE USER TO GIVE A COMMAND. AFTER READING IT, IT WILL BE INTERPRETED TO DETERMINE WHAT ACTION HE DESIRES. THE COMMAND SYNTAX WILL BE DEALT WITH ONLY TO THE LEVEL NECESSARY TO DETERMINE THE BASIC INTENT AND SEPARATE OUT ANY INFORMATION (E.G. ARGUMENTS) FOR THE IPAD UTILITY.

F.B DE-ACTIVATE SUBTASK STEP

THIS IS AN ALTERNATE ENTRY POINT TO BE USED WHEN A PAUSE HAS BEEN GIVEN, FOLLOWED BY A COMMAND OTHER THAN GO. A PUSH-DOWN STACK WILL HAVE TO BE KEPT TO INSURE A LAST-IN-FIRST-OUT PROCESSING ORDER.

F.C RE-ACTIVATE SUBTASK STEP

THE PUSH-DOWN STACK OF INTERRUPTED SUBTASK STEPS MUST BE INTERROGATED TO LOCATE THE STEP WHICH IS TO BE ACTIVATED.

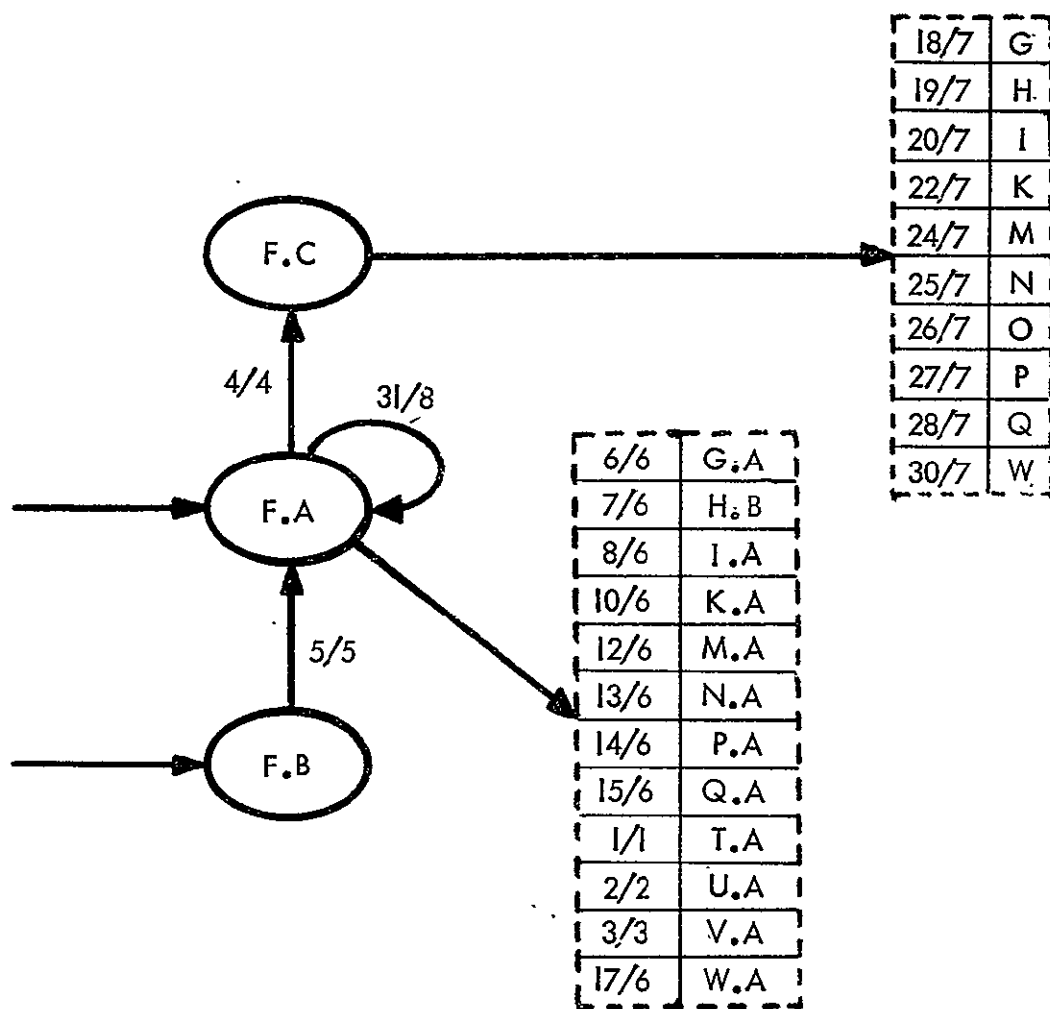
F : SUBTASK COMMAND MODE
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P.F.A	F.A	31	8
	F.C	4	4
	P.G.A	6	6
	P.H.A	7	6
	P.I.A	8	6
	P.K.A	10	6
	P.M.A	12	6
	P.N.A	13	6
	P.P.A	14	6
	P.Q.A	15	6
	P.T.A	16	1
	P.U.A	2	2
	P.V.A	3	3
	P.W.A	17	6
P.F.B	F.A	5	5
P.F.C *	P.G.A	18	7
	P.H	19	7
	P.I	20	7
	P.K	22	7
	P.M	24	7
	P.N	25	7
	P.O	26	7
	P.P	27	7
	P.Q	28	7
	P.W	30	7

* Since these are transitions to interrupted states, the
node names at level 2 cannot be specified.

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 2 TRANSITION DIAGRAM

STATE F: SUBTASK COMMAND MODE

F : SUBTASK COMMAND MODE
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	STOP
2	QUIT
3	TERMINATE
4	RETURN
5	RECOVERY INFORMATION FROM SUBTASK STEP ROLLOUT RECORD, OLD PUSH-DOWN STACK
6	HELP
7	SEARCH
8	ENTER DATA
11	MODIFY DATA
12	CONSTRUCT JOB
13	EXECUTE
14	DISPLAY
15	DISPOSE
17	DEFINE
18	PUSH-DOWN STACK WITH STATE G ON TOP
19	PUSH-DOWN STACK WITH STATE H ON TOP
20	PUSH-DOWN STACK WITH STATE I ON TOP
21	PUSH-DOWN STACK WITH STATE J ON TOP
22	PUSH-DOWN STACK WITH STATE K ON TOP
23	PUSH-DOWN STACK WITH STATE L ON TOP
24	PUSH-DOWN STACK WITH STATE M ON TOP
25	PUSH-DOWN STACK WITH STATE N ON TOP
26	PUSH-DOWN STACK WITH STATE O ON TOP
27	PUSH-DOWN STACK WITH STATE P ON TOP
28	PUSH-DOWN STACK WITH STATE Q ON TOP
30	PUSH-DOWN STACK WITH STATE W ON TOP
31	INCORRECT COMMAND FORMAT

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	POINTERS TO SUBTASK STEP TO BE TERMINATED
2	-
3	-
4	PUSH-DOWN STACK
5	MODIFIED PUSH-DOWN STACK, AND THE DE-ACTIVATED STS

F : SUBTASK COMMAND MODE
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** OUTPUT / RESULT LIST (CONTINUED) *****

NUMBER	TEXT
6	COMMAND BROKEN UP INTO THE BASIC COMPONENTS
7	LAST RECORDED LINE SENT TO THE TERMINAL BEFORE THE INTERRUPTION IS RE-ISSUED TO THE TERMINAL; ALSO THE MODIFIED PUSH DOWN STACK.
8	EXPLANATION OF THE ERROR, REQUEST FOR RETRY

***** CROSS REFERENCED TRANSITIONS *****

STATE IS ACCESSIBLE FROM

F.A	E.3
	E.5
	G.E
	G.I
	H.C
	H.D
	H.E
	I.D
	K.D
	M.A
	M.C
	F.C
	P.J
	P.E
	Q.C
	Q.J
	Q.E
	Q.F
	Q.G
	Q.H
	Q.I
	Q.J
	Q.K
	T.B
	W.C

LEVEL 2
COMPONENTS OF STATE G
LEARNING ABOUT IPAD

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

G.A VALIDATE USER

THE USER MUST BE VALIDATED FOR THE PROGRAMMED COURSE OF INSTRUCTION HE WANTS TO BEGIN OR CONTINUE. COMPLETE COURSES COVERING DIFFERENT SUBJECTS WILL BE OFFERED. A PARTICULAR SUBJECT MAY BE COVERED AT SEVERAL LEVELS OF DETAIL.

G.B RETRIEVE STUDENT RECORD

USER PROFILE INFORMATION IS MAINTAINED IN THE SYSTEM SECURITY FILE. A RECORD IS KEPT OF EACH USERS LEVEL OF PROFICIENCY BASED ON GRADES FOR COURSES COMPLETED AND HIS DYNAMIC USE OF THE TEACHING FACILITY WHILE HE WORKS.

G.C ESTABLISH STUDENT RECORD

IF THIS IS A FIRST REQUEST FOR HELP OR FOR A PROGRAMMED COURSE A STUDENT RECORD IS ESTABLISHED FOR THE USER.

G.D RETRIEVE LESSON PLAN

THE SCENARIO FOR THE PROPER LESSON IS OBTAINED FOR USE IN PRESENTING THE MATERIAL TO THE USER.

G.E PRESENT LESSON

THE MATERIAL IS PRESENTED TO THE USER AT A RATE DETERMINED BY THE USERS ABILITY TO LEARN.

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
-------	--------------------

G.F	DETERMINE CONTEXT
-----	-------------------

THE USER HAS MADE A REQUEST FOR HELP. HE IS EITHER BETWEEN ACTIVITIES OR HE HAS INTERRUPTED HIMSELF TO GET ASSISTANCE. IN THE LATTER CASE THE SYSTEM WILL ATTEMPT TO DETERMINE WHAT TYPE OF SCENARIO IS MOST LIKELY TO SATISFY HIS NEEDS WITHOUT BEING TOLD DIRECTLY. IF THE USER IS BETWEEN ACTIVITIES OR THE CONTEXT OF HIS PREVIOUS ACTIVITY DOES NOT PROVIDE A GOOD GUESS, THE SYSTEM AND USER ENGAGE IN A DIALOGUE TO DETERMINE THE TYPE OF HELP HE WANTS.

G.G	RETRIEVE SCENARIO
-----	-------------------

A SCENARIO TO GUIDE A HELP SESSION IS RETRIEVED. THIS HELP IS NOT PROGRAMMED TEACHING BUT QUERY-ANSWER, DISPLAYS OF OPTIONS, ETC.

G.H	SELECT LANGUAGE LEVEL
-----	-----------------------

THE USER RECORD IS USED TO SELECT A LANGUAGE LEVEL COMPATIBLE WITH THE USERS PROFICIENCY. THE USER MAY CHANGE LANGUAGE LEVEL AT ANY TIME.

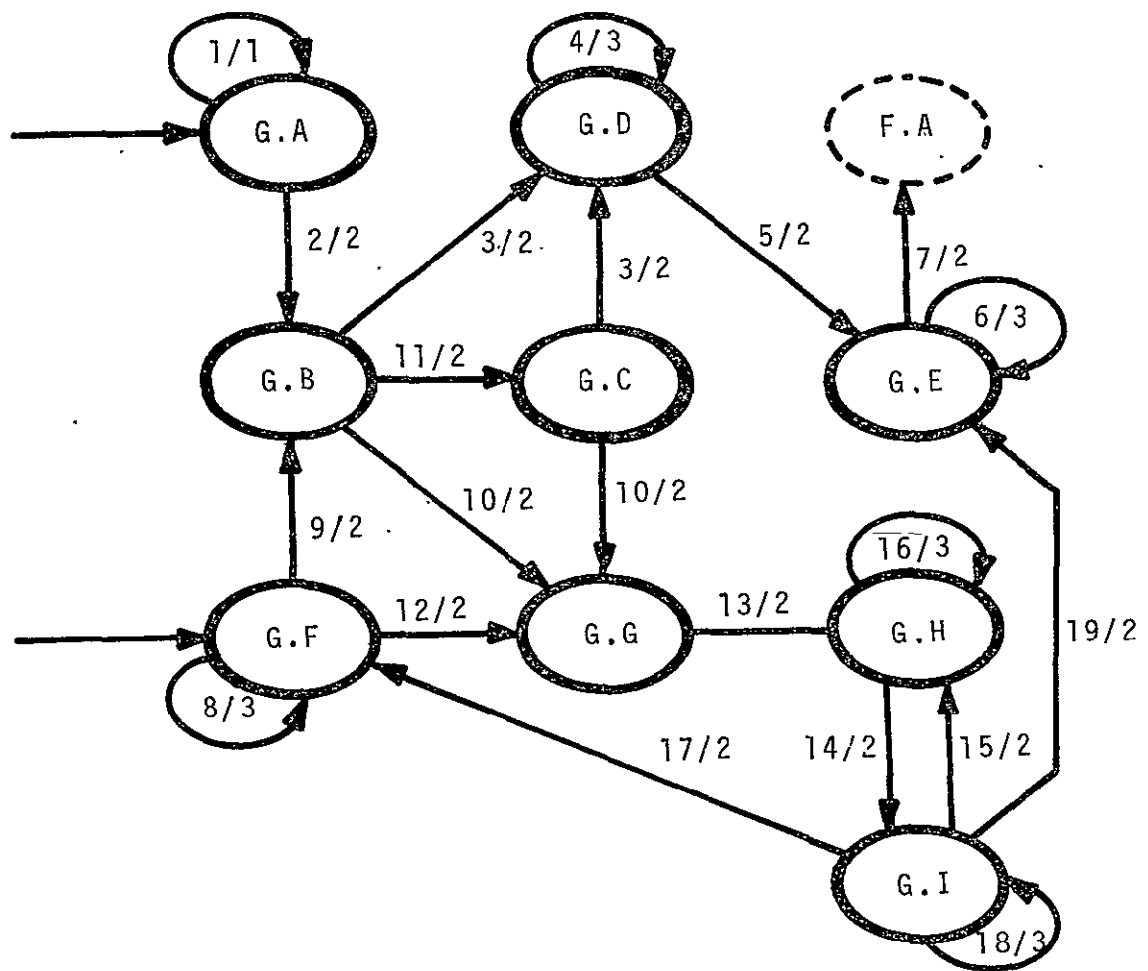
G.I	RESPOND TO USER QUERIES
-----	-------------------------

THE HELP SESSION IS A DIALOGUE BETWEEN THE USER AND THE SYSTEM.

G : LEARNING ABOUT IPAD
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P G.A	G.A	1	1
	G.B	2	2
G.B	G.C	11	2
	G.D	3	2
	G.G	13	2
G.C	G.D	3	2
	G.G	13	2
G.D	G.D	4	3
	G.E	5	2
G.E	P F.A	7	2
	G.E	6	3
P G.F	G.B	9	2
	G.F	5	3
	G.G	12	2
G.G	G.H	13	2
G.H	G.H	16	3
	G.I	14	2
G.I	P F.A	19	2
	G.F	17	2
	G.H	15	2
	G.I	18	3



LEVEL 2 TRANSITION DIAGRAM

STATE G: LEARNING ABOUT IPAD

ORIGINAL PAGE IS
OF POOR QUALITY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	USER NOT VALIDATED FOR PROGRAMMED COURSE
2	USER VALIDATED
3	USER TRAINING RECORDS AVAILABLE
4	USER/SYSTEM DIALOGUE RELATIVE TO LESSON SELECTION INCOMPLETE
5	LESSON SCENARIO IN USER WORKING AREA
6	LESSON SESSION INCOMPLETE
7	LESSON SESSION COMPLETE
8	MORE INFORMATION REQUIRED TO DETERMINE TYPE OF HELP WANTED
9	INITIAL HELP CONTEXT DETERMINED
10	USER PROFICIENCY DATA AVAILABLE
11	SET UP NEW USER USER TRAINING RECORD
12	SECOND AND LATER HELP CONTEXT DETERMINED
13	HELP SCENARIO RETRIEVED SUITABLE FOR SELECTED CONTEXT
14	LANGUAGE LEVEL SELECTED
15	USER DESIRES CHANGE IN LANGUAGE LEVEL
16	USER/SYSTEM DIALOGUE RELATIVE TO LANGUAGE CHANGE INCOMPLETE
17	USER WANTS TO CHANGE HELP CONTEXT
18	USER/SYSTEM HELP DIALOGUE INCOMPLETE
19	HELP COMPLETE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE TO SELECT ANOTHER COURSE OR TERMINATE
2	-
3	MESSAGE TO USER INFORMING HIM TO PROCEED

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
G.A	F.A

LEVEL 2
COMPONENTS OF STATE H
SEARCHING THROUGH THE LIBRARIES

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

H.3 INTERPRET COMMAND

THE USER MAY WISH TO CONTROL HIS OWN SEARCH BY SCANNING DICTIONARY AND DIRECTORY ENTRIES TO IDENTIFY LIBRARY ENTRIES TO BE DISPLAYED. HE MAY ALSO WANT THE SYSTEM TO PERFORM SEARCHES UTILIZING SELECTION CRITERIA HE SUPPLIES.

H.C USER CONTROLLED SEARCH

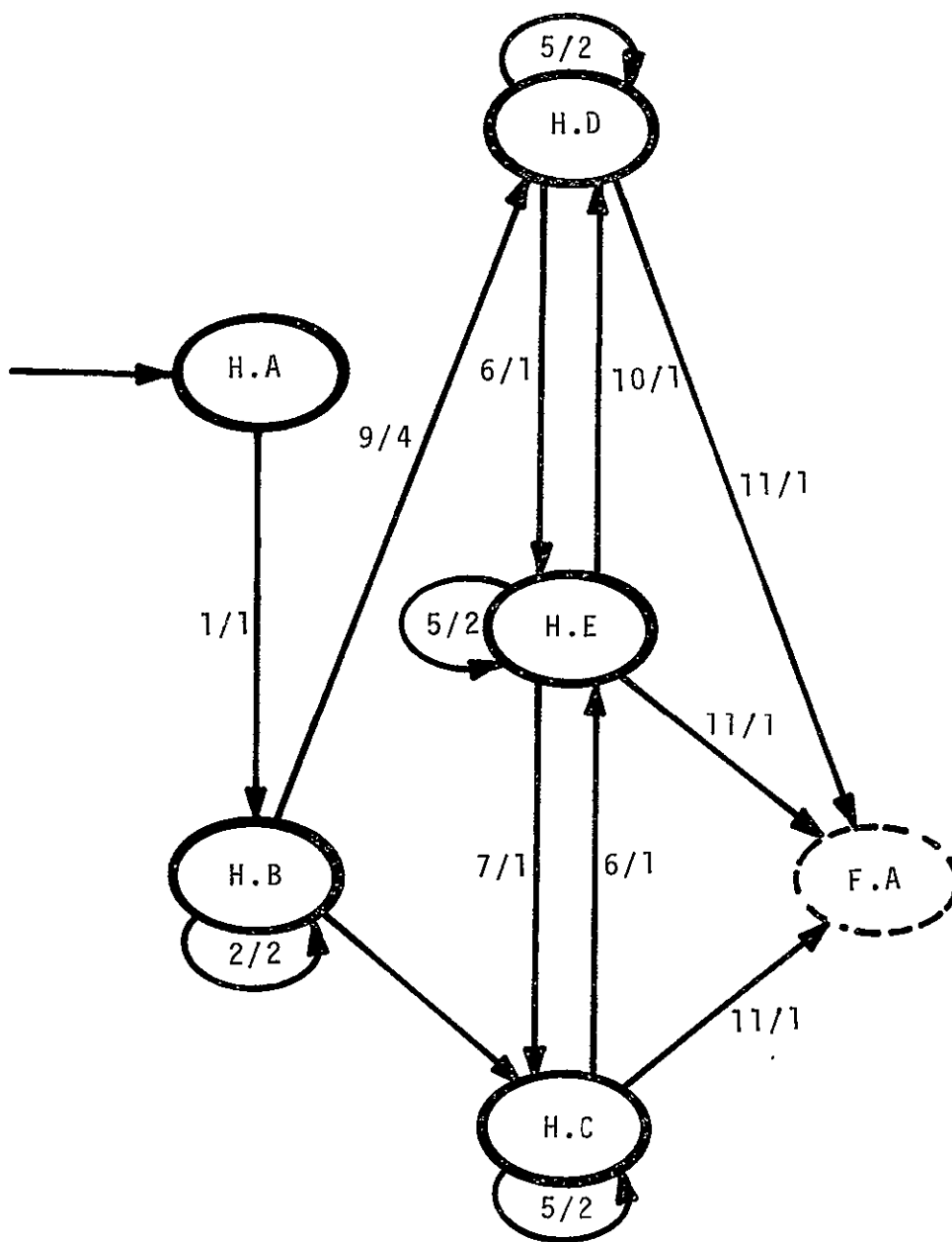
A SEARCH FOR A SPECIFIC DICTIONARY, DIRECTORY, OR LIBRARY ENTRY MAY BE MADE, OR THE USER MAY PAGE THROUGH AN ENTIRE DICTIONARY OR DIRECTORY.

H.J SYSTEM CONTROLLED SEARCH

AN INFORMATION SELECTION EXPRESSION IS GIVEN TO THE SYSTEM AND USED TO CONTROL THE SEARCH.

H.E DISPLAY SELECTED INFORMATION

INFORMATION IDENTIFIED BY A SEARCH IS DISPLAYED TO THE USER FOLLOWING VALIDATION FOR READ ACCESS.



LEVEL 2 TRANSITION DIAGRAM

STATE H: SEARCHING THROUGH THE LIBRARIES

H : SEARCHING THROUGH THE LIBRARIES
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→H.B	H.B	2	2
	H.C	4	4
	H.D	9	4
H.C	→F.A	11	1
	H.C	5	2
	H.E	6	1
H.D	→F.A	11	1
	H.D	5	2
	H.E	6	1
H.E	→F.A	11	1
	H.C	7	1
	H.D	10	1
	H.E	5	2

H : SEARCHING THROUGH THE LIBRARIES (CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
2	MORE INFORMATION REQUIRED TO COMPLETE COMMAND ANALYSIS
4	COMMAND ANALYSIS COMPLETE, USER CONTROLLED SEARCH DESIRED.
5	ADDITIONAL USER INPUT REQUIRED
6	DATA FOR DISPLAY LOCATED
7	USER SIGNAL TO START NEW USER CONTROLLED SEARCH
3	COMMAND ANALYSIS COMPLETE, SYSTEM CONTROLLED SEARCH DESIRED.
10	USER SIGNAL TO START NEW SYSTEM CONTROLLED SEARCH
11	USER SIGNAL THAT HE HAS COMPLETED HIS ACTIVITY

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE INFORMING USER TO PROCEED
2	MESSAGE REQUESTING USER TO ENTER MORE INFORMATION
4	PARSED COMMAND AND COMMAND CONTROL TABLE

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
H.3	F.A F.A.D

LEVEL 2
COMPONENTS OF STATE I
CREATING LIBRARY ENTRIES

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.A INTERPRET COMMAND

THE USER MAY ENTER DATA TO BUILD A NEW LIBRARY ENTRY. THIS MAY BE AN INSTANCE OF A SYSTEM DATA STRUCTURE SUCH AS A CODING MODULE OR A STORED DATA DEFINITION. THE DATA ENTERED MAY ALSO BE VALUES WHICH COMPRISE AN INSTANCE OF A USER DEFINED DATA SET.

I.B VALIDATE USER

THE USER MUST HAVE PERMISSION TO ENTER PARTICULAR TYPES OF DATA INTO THE CL OR HIS STL.

I.C CONSTRUCT LIBRARY ENTRY

A COMPLETE, NEW LIBRARY ENTRY(DIRECTORY AND TEXT) IS CONSTRUCTED. A DICTIONARY ENTRY, IF REQUIRED, IS ALSO MADE.

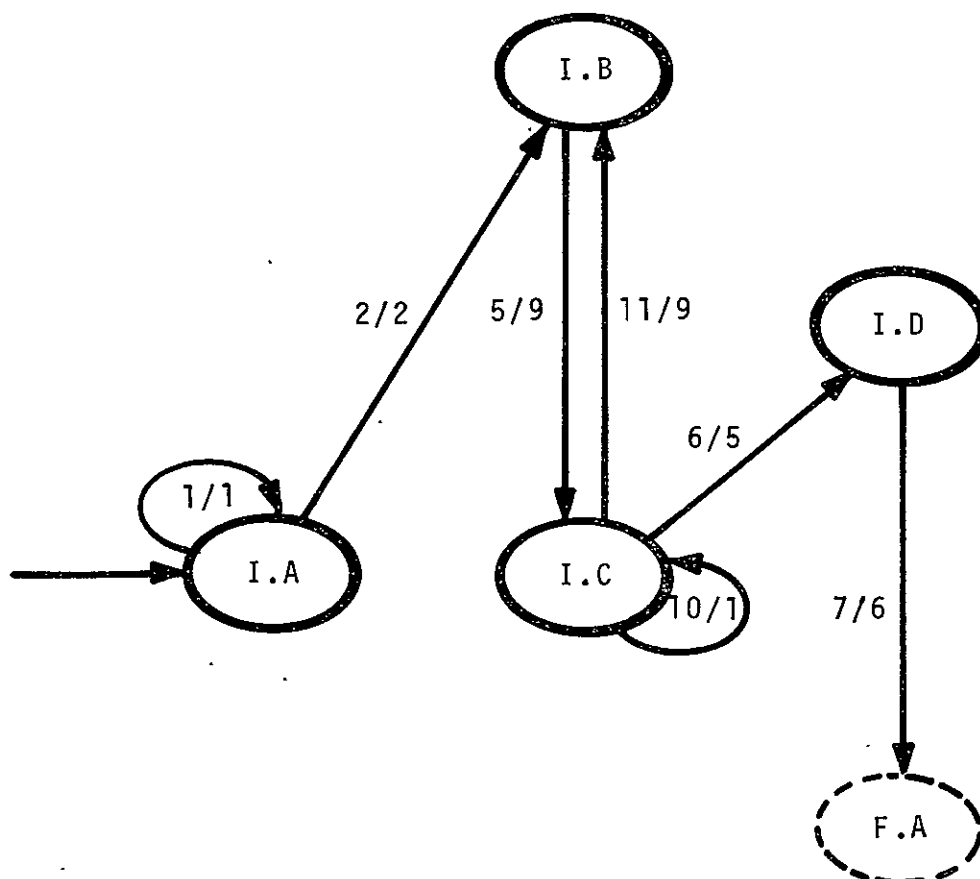
I.D DISCONNECT USER FROM DATA

I : CREATING LIBRARY ENTRIES
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (# = ENTRY)	TO STATE (# = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
1.A	1.A	1	1
	1.B	2	2
1.3	1.B	4	4
	1.C	5	9
1.3	1.B	11	11
	1.C	13	1
	1.D	6	5
1.J	1.F.A	7	6

TRANSITION DIAGRAM



I : CREATING LIBRARY ENTRIES
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	MORE INFORMATION REQUIRED TO COMPLETE COMMAND ANALYSIS
2	COMMAND ANALYSIS COMPLETE
4	USER NOT PERMITTED REQUESTED ACTION
5	USER VALIDATED FOR REQUESTED ACTION
6	LIBRARY ENTRY CONSTRUCTION COMPLETE
7	USER DISCONNECTED FROM LE
10	LIBRARY ENTRY CONSTRUCTION INCOMPLETE
11	ADDITIONAL VALIDATION REQUIRED FOR DERIVATIVE ACTIVITY

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE REQUESTING USER TO SUPPLY MORE INFORMATION
2	PARSED COMMAND AND COMMAND CONTROL TABLE
4	MESSAGE INFORMING USER OF LACK OF VALIDATION. ASK FOR ALTERNATE REQUEST FROM HIM.
5	LE IN USER WORKING AREA
6	LE AVAILABLE IN DATA BASE
3	-
10	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
I.A	F.A

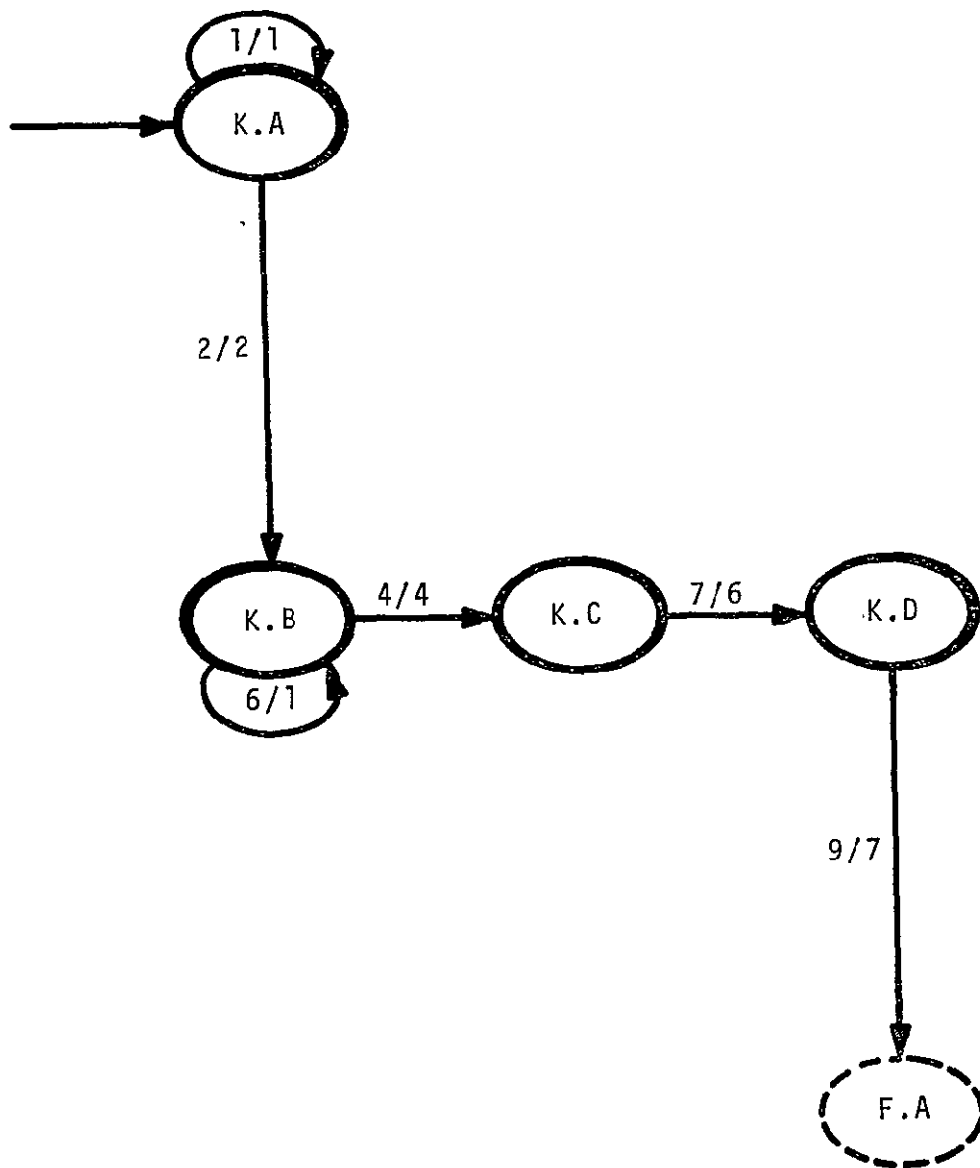
LEVEL 2
COMPONENTS OF STATE K
MODIFYING LIBRARY ENTRIES

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
K.A	CONNECT USER WITH DATA TO BE MODIFIED
K.B	PERFORM MODIFICATIONS WITH DIALOG
K.C	UPDATE DIRECTORY ENTRY
K.D	DISCONNECT USER FROM DATA

***** ALLOWED TRANSITIONS *****

FROM STATE (# = ENTRY)	TO STATE (# = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.A	K.A	1	1
	K.B	2	2
K.B	K.B	6	1
	K.C	4	4
K.C	K.D	7	6
K.D	→F.A	9	7



LEVEL 2 TRANSITION DIAGRAM

STATE K: MODIFYING LIBRARY ENTRIES

K : MODIFYING LIBRARY ENTRIES
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	IDENTIFYING AND LOCATING INFORMATION INCOMPLETE
2	LIBRARY ENTRY TO BE MODIFIED EXISTS AND USER IS VALID- ATED TO PERFORM MODIFICATIONS
4	MODIFICATION COMPLETED
6	MODIFICATIONS INCOMPLETE
7	DIRECTORY ENTRY UPDATE COMPLETE
9	USER IS DISCONNECTED FROM DATA

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE ISSUED TO USER REQUESTING ADDITIONAL INFORM- ATION
2	USER IS CONNECTED TO LIBRARY ENTRY
4	UPDATED LE TEXT IN USER AREA
6	UPDATED LE ATTACHED TO USER
7	UPDATED LE AVAILABLE IN DATA BASE

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
K.A	F.A

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 2
COMPONENTS OF STATE M
CONSTRUCTING A JOB

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

M.A DETERMINE AVAILABLE JOB COMPONENTS

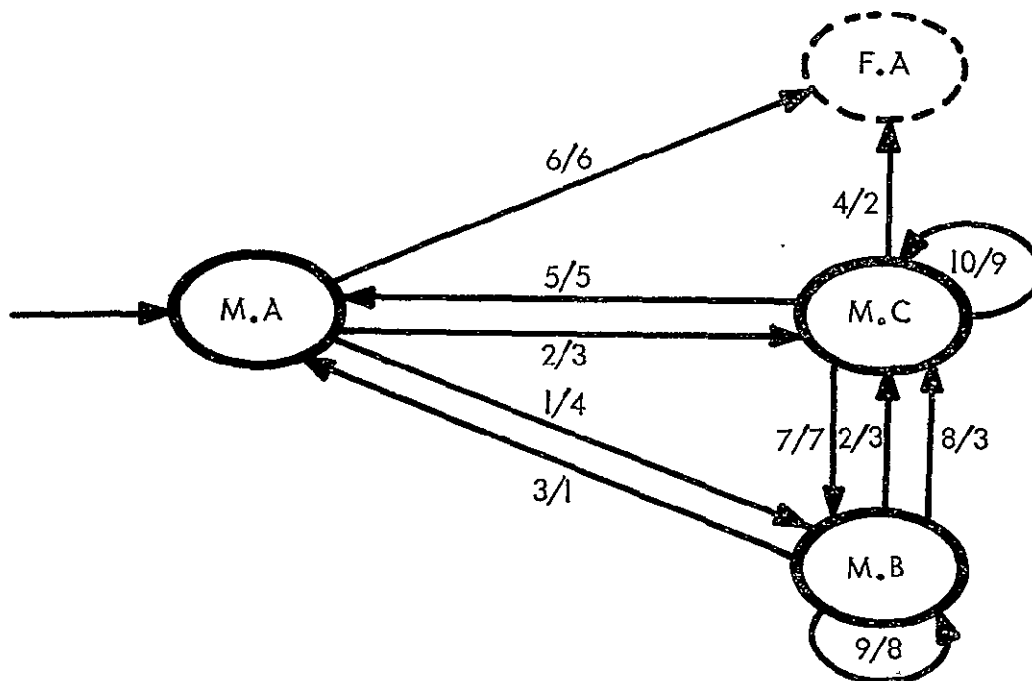
TAKE THE USERS LIST OF OMS AND FIND OUT HOW MANY
ARE CURRENTLY DEFINED AND ACCESSABLE AND HOW MANY ARE
YET TO BE DEFINED. THE USER MAY THEN CHOOSE TO
CONSTRUCT THE OMS OR NOT

M.B CONSTRUCT AN OM LIBRARY ENTRY

M.C CONSTRUCT A JOB LIBRARY ENTRY

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
M.A	F.A	6	6
	M.B	1	4
	M.C	2	3
M.B	M.A	3	1
	M.B	3	8
	M.C	2	3
M.C	M.C	3	3
	F.A	4	2
	M.A	5	5
	M.B	7	7
	M.C	13	9



LEVEL 2 TRANSITION DIAGRAM

STATE M : CONSTRUCTING A JOB

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	OM DIRECTORY INDICATING THAT AT LEAST ONE REQUIRED OM IS UNDEFINED AND/OR INACCESSABLE AND AN INDICATION FROM THE USER THAT HE DESIRES TO CONSTRUCT THE OM(S)
2	OM DIRECTORY INDICATING THAT ALL REQUIRED OMS ARE DEFINED AND ACCESSABLE.
3	NON-EMPTY LIST OF OMS TO BE DEFINED
4	EMPTY LIST OF JOBS TO BE DEFINED
5	NON-EMPTY LIST OF JOBS TO BE DEFINED
6	OM DIRECTORY INDICATING THAT AT LEAST ONE REQUIRED OM IS UNDEFINED AND/OR INACCESSABLE AND AN INDICATION FROM THE USER THAT HE DOES NOT WANT TO CONSTRUCT THE OM.
7	USER'S INDICATION THAT ONE OR MORE OMS MUST BE DEFINED
8	ENTRY FLAG FROM M.C AND ALSO INPUT NO. 2
9	OM CONSTRUCTION INFORMATION WHICH MUST COME FROM THE USER
10	JOB CONSTRUCTION INFORMATION WHICH MUST COME FROM THE USER

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	ONE OR MORE NEWLY DEFINED OMS IN THE STL AND THE LIST OF UNDEFINED OMS
2	JOB DEFINITIONS COMPLETED IN THE STL
3	COMPLETE LIST OF OM NAMES BY LIBRARY
4	LIST OF NAMES FOUND (BY LIBRARY), AND THOSE YET TO BE FOUND
5	LIST OF JOBS YET TO BE DEFINED
6	-
7	OM NAME(S)
8	PROMPTS FROM THE SYSTEM FOR THE PROPER OM CONSTRUCTION INFORMATION
9	PROMPTS FROM THE SYSTEM FOR THE PROPER JOB CONSTRUCTION INFORMATION

***** CROSS REFERENCED TRANSITIONS *****

STATE IS ACCESSIBLE FROM

M.A

F.A

LEVEL 2
COMPONENTS OF STATE N
EXECUTING A JOB

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

N.A ESTABLISH THE REQUIRED LEN LIST

SCAN THE JOB SPECIFICATIONS FOR ALL INPUT/OUTPUT
LE, AND WITH THE QUALIFYING INFORMATION GIVEN WITH THE
EXECUTION COMMAND, ESTABLISH THE LIST OF NAMES FOR
SEARCHING IN THE LIBRARIES.

N.B CHECK FOR LEN IN LIBRARIES

LOOK FOR THE LEN IN THE STL AND THEN IN THE CL.

N.C PREPARE JOB FOR EXECUTION

THE SKELETON OF THE JOB DEFINITION MUST NOW BE
FILLED IN WITH ITEMS PERTINENT TO THIS EXECUTION. THE
EXECUTABLE CODE FILES MUST BE SET UP PROPERLY, AND THE
CONTROL CARDS FOR THIS EXECUTION MUST BE GENERATED.

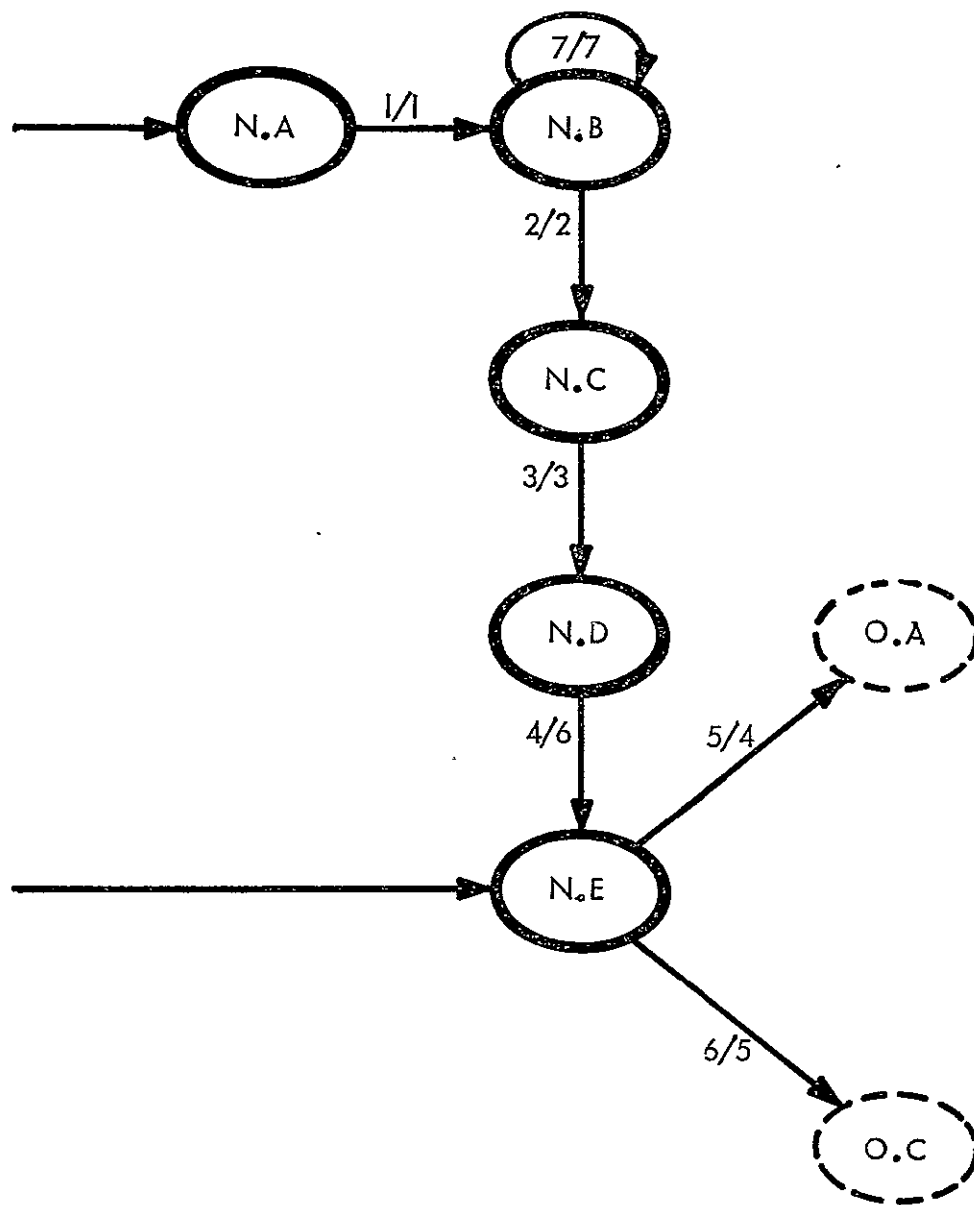
N.D INITIATE EXECUTION

N.E SUBTASK STEP EXECUTING

THIS REPRESENTS THE STATE OF THE SYSTEM WHEN THE
JOB IS EXECUTING AND NOT COMMUNICATING WITH THE USER.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P.N.A	N.B	1	1
N.B	N.C	7	7
	N.C	2	2
N.C	N.D	3	3
N.D	N.E	4	6
P.N.E	P.O.A	5	4
	P.O.C	6	5



LEVEL 2 TRANSITION DIAGRAM
STATE N : EXECUTING A JOB

N : EXECUTING A JOB
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	LIST OF ULEN FROM JOB DESCRIPTION AND QUALIFYING INFORMATION FROM THE EXECUTION REQUEST.
2	STL, CL DIRECTORIES CONTAINING ALL REQUIRED LEN WITH PROPER ACCESS CODES
3	STATUS COMPLETE ON ALL JOB COMPONENT FILES
4	INDICATOR FROM THE OS THAT THE STS IS EXECUTING
5	INPUT REQUEST COMMAND FROM THE STS
6	OUTPUT COMMAND FROM THE STS
7	STL, CL DIRECTORIES LACKING SOME OF THE LEN

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	LIST OF ALL LEN REQUIRED FOR THE JOB
2	LINKAGE ESTABLISHED TO ALL REQUIRED LE
3	EXECUTABLE CODE FILE(S), CONTROL CARD STREAM
4	TERMINAL INPUT REQUEST
5	TERMINAL OUTPUT REQUEST
6	-
7	EXPLANATION TO THE USER, REQUEST FOR ADDITIONAL INFORMATION TO USE IN LOCATING THE MISSING LEN

***** CROSS REFERENCE) TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.A	F.A
N.E	O.C

LEVEL 2
COMPONENTS OF STATE 0
COMMUNICATING WITH A JOB

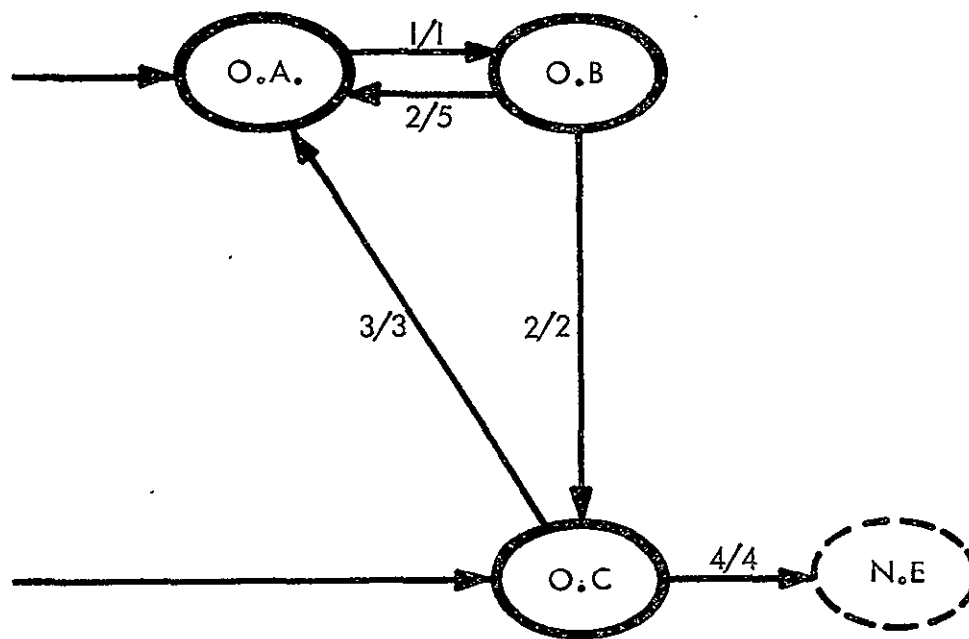
ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
0.4	SUBTASK STEP REQUESTING USER INPUT
0.3	SUBTASK STEP PROCESSING USER INPUT WORK NECESSARY TO INTERPRET THE INPUT AND CORRECT ANY ERRORS.
0.0	SUBTASK STEP OUTPUTTING INFORMATION

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P0.4	0.3	1	1
0.3	0.4	2	2
	0.0	2	2
P0.0	P0.4	4	4
	0.4	3	3



LEVEL 2 TRANSITION DIAGRAM

STATE O : COMMUNICATING WITH A JOB

D : COMMUNICATING WITH A JOB
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	USER INPUT LINE TO STS
2	INPUT COMMAND FROM STS
3	END OF OUTPUT INDICATOR
4	COMPLETED INPUT FROM USER

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	TERMINAL LINE
2	REQUEST FOR MORE INPUT
3	1 OR MORE LINES TO TERMINAL, INPUT REQUEST
4	OPTIONAL ACKNOWLEDGEMENT OF INPUT RECEIVED CORRECTLY
5	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
0.A	N.E N.E.A
0.C	N.E N.E.A

C-3

LEVEL 2
COMPONENTS OF STATE P
DISPLAYING RESULTS

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

P.A INTERPRFT DISPLAY REQUEST

ANALYZE THE DISPLAY REQUEST TO DETERMINE THE LE
AND LV INVOLVED AND REQUEST THE SELECTION CRITERION

P.B EVALUATE SELECTION CRITERIA

ANALYZE THE CRITERIA FOR SELECTING DATA TO BE
DISPLAYED AND FORM THE ANALYTICAL EXPRESSION

P.C ESTABLISH SUPER-SET INFORMATION

FETCH INFORMATION REQUIRED FOR THE SELECTION AND
IF THE SELECTION EXPRESSION IS NOT IN THE TERMS OF THE
RAW INFORMATION, TRANSFORM IT PRIOR TO APPLYING THE
SELECTION CRITERIA

P.D SELECT PROPER SUB-SET INFORMATION

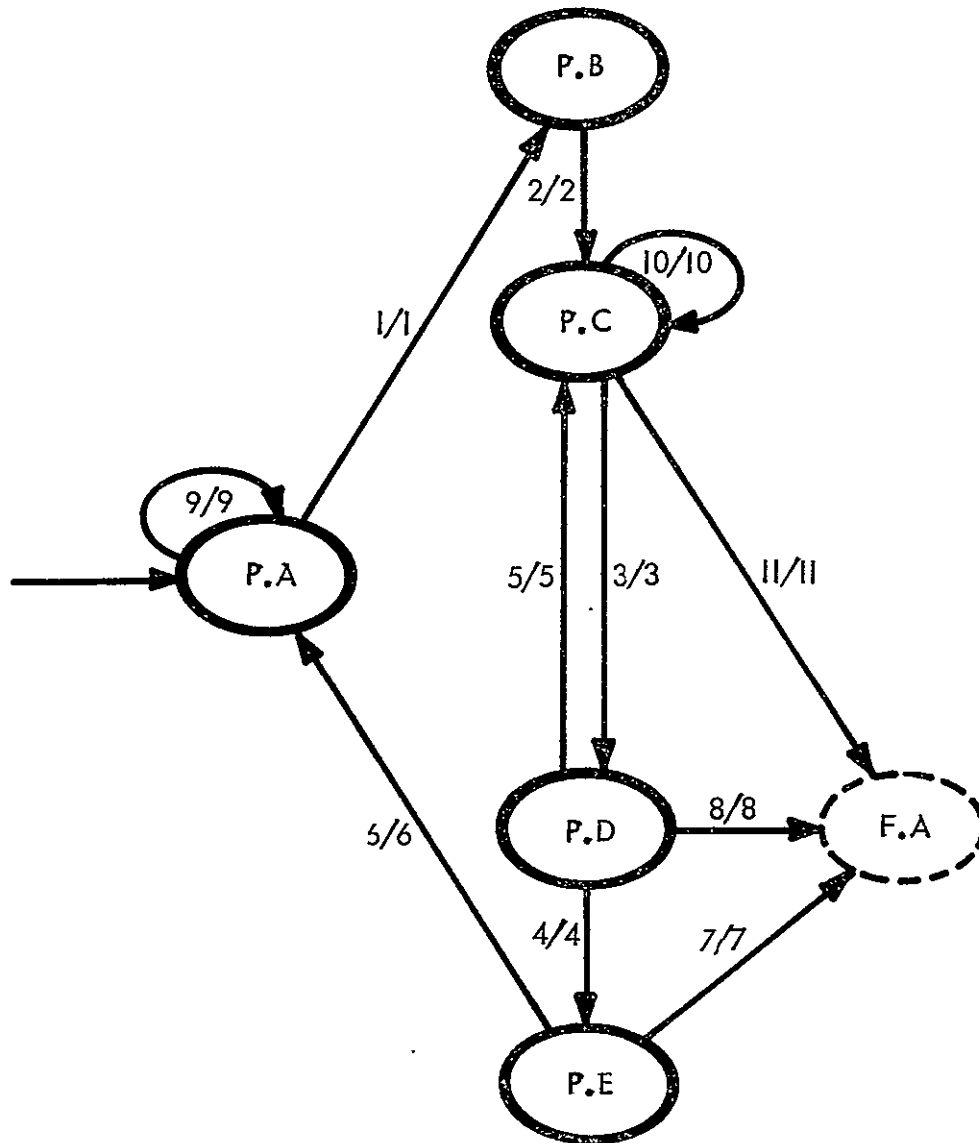
APPLY THE SELECTION CRITERIA TO THE SUPER-SET
INFORMATION TO ESTABLISH THE DESIRED SUBSET INFORMATION

P.E DISPLAY REQUESTED INFORMATION

P : DISPLAYING RESULTS
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P.A	P.A	9	9
	P.B	1	1
P.B	P.C	2	2
P.C	P.F.A	11	11
	P.C	13	13
	P.D	3	3
P.J	P.F.A	5	5
	P.C	5	5
	P.E	4	4
P.E	P.F.A	7	7
	P.A	6	6



LEVEL 2 TRANSITION DIAGRAM

STATE P : DISPLAYING RESULTS

P : DISPLAYING RESULTS
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	PROPERLY FORMATTED DISPLAY REQUEST
2	SUFFICIENT INFORMATION TO ESTABLISH A SELECTION CRITERIA
3	PART OR ALL OF THE SUPERS _{ET} INFORMATION FROM THE - LIBRARY
4	EMPTY SEARCH TABLE
5	NON-EMPTY SEARCH TABLE
6	USER REQUEST FOR MORE DISPLAY
7	USER TERMINATION CODE
8	USER INDICATOR THAT HE DOES NOT WANT THE INFORMATION DISPLAYED (THIS IMPLIES THAT HE IS ONLY INTERESTED IN KNOWING THAT THE INFORMATION EXISTS)
9	IMPROPER OR INSUFFICIENT INFORMATION FOR THE SELECTION CRITERIA
10	LEN REQUIRED FOR THE SEARCH ARE NOT ACCESSABLE BY THIS USER
11	USERS COMMAND TO EXIT

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	FORMAL EXPRESSION OF DISPLAY REQUEST
2	FORMAL EXPRESSION OF SELECTION CRITERION
3	SUPER-SET INFORMATION
4	INFORMATION TO BE DISPLAYED
5	REMAINING SEARCH LIST
6	-
7	-
8	-
9	EXPLANATION OF ERROR, REQUEST FOR CORRECTION/ADDITION
10	ERROR MESSAGE TO USER, WAIT FOR A RESOLUTION OR USERS COMMAND TO EXIT
11	-

***** CROSS REFERENCE J. TRANSITIONS *****

STATE IS ACCESSIBLE FROM

P.A	F.A
	F.A.D

LEVEL 2
COMPONENTS OF STATE Q
DISPOSITION OF LIBRARY ENTRIES

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

Q.A INTERPRET COMMAND

THE COMMAND IS ANALYZED TO DETERMINE WHICH TYPE OF ACTION IS TO BE PERFORMED. LIBRARY ENTRIES MAY BE PURGED, MOVED TO AND FROM ARCHIVE STORAGE, MOVED FROM STL TO STL, STL TO CL, CL TO STL, AND FROM CL OR STL TO A DESTINATION OUTSIDE OF IPAD.

Q.B VALIDATE USER FOR ACTIVITY

VALIDATION REQUIRES PERMISSION TO PERFORM DESIRED ACTION AS WELL AS PERMISSION TO ACCESS THE DATA WHICH IS REFERENCED.

ACTIVITY VALIDATION FOR THE ACTIVE USER IS DONE HERE. ADDITIONAL VALIDATION MAY BE REQUIRED FOR THE RECIPIENT OF DATA WHICH IS MOVED FROM AN STL TO ANOTHER STL, OR TO A NON-IPAD DESTINATION. THE SAME IS TRUE FOR DATA RESIDING IN THE CL. A SPECIAL CLASS OF NON-IPAD USERS ELIGIBLE TO RECEIVE IPAD CONTROLLED DATA HAVE VALIDATION INFORMATION IN THE SYSTEM SECURITY FILE.

Q.C PURGE A CL ENTRY

Q.D PURGE A STL ENTRY

Q.E MOVE FROM ARCHIVE TO CL

Q.F MOVE FROM CL TO ARCHIVE

Q.G MOVE FROM STL TO CL

Q.H MOVE FROM CL TO STL

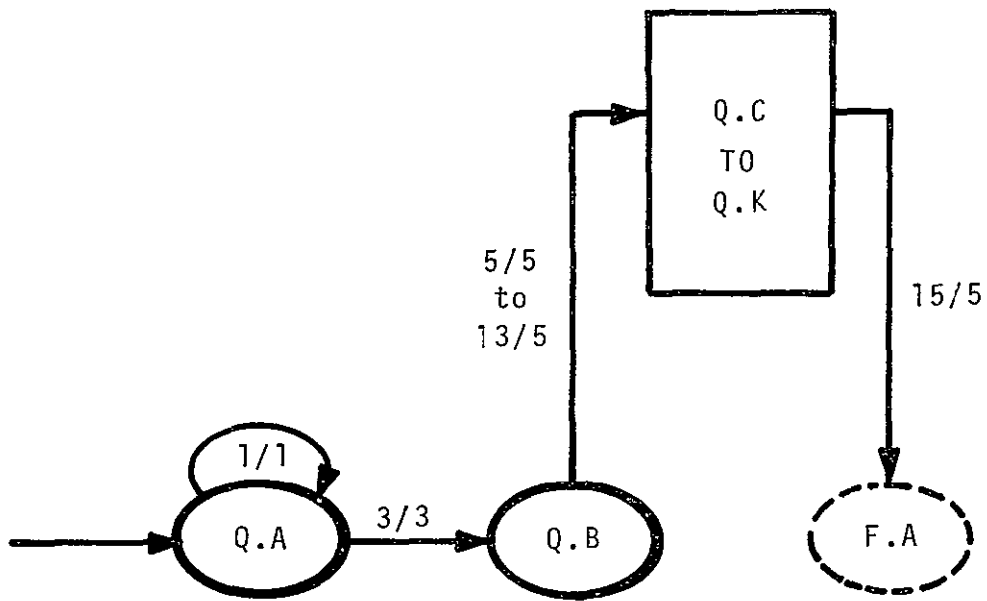
Q : DISPOSITION OF LIBRARY ENTRIES
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE	LONG NAME AND TEXT
Q.I	MOVE FROM STL TO STL
Q.J	MOVE FROM STL OUT OF IPAD
Q.K	MOVE FROM CL OUT OF IPAD

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
Q.A	Q.A	1	1
	Q.B	3	3
Q.B	Q.C	5	5
	Q.D	6	5
	Q.E	7	5
	Q.F	8	5
	Q.G	9	5
	Q.H	10	5
	Q.I	11	5
	Q.J	12	5
	Q.K	13	5
Q.C	Q.F.A	15	5
Q.D	Q.F.A	15	5
Q.E	Q.F.A	15	5
Q.F	Q.F.A	15	5
Q.G	Q.F.A	15	5
Q.H	Q.F.A	15	5
Q.I	Q.F.A	15	5
Q.J	Q.F.A	15	5
Q.K	Q.F.A	15	5



LEVEL 2 TRANSITION DIAGRAM

STATE Q: DISPOSITION OF LIBRARY ENTRIES

Q : DISPOSITION OF LIBRARY ENTRIES
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	MORE INFORMATION REQUIRED TO COMPLETE COMMAND ANALYSIS
3	COMMAND ANALYSIS COMPLETE
5	USER VALIDATED FOR CL PURGE
6	USER VALIDATED FOR STL PURGE
7	USER VALIDATED FOR MOVE ARCHIVE TO CL
8	USER VALIDATED FOR MOVE CL TO ARCHIVE
9	USER VALIDATED FOR MOVE STL TO STL
10	USER VALIDATED FOR MOVE CL TO STL
11	USER VALIDATED FOR MOVE STL TO STL
12	USER VALIDATED FOR MOVE STL OUT OF IPAD
13	USER VALIDATED FOR MOVE CL OUT OF IPAD
15	DISPOSITION COMPLETE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE REQUESTING USER TO ENTER MORE INFORMATION
3	PARSED COMMAND AND COMMAND CONTROL TABLE
5	MESSAGE INFORMING USER TO PROCEED

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
Q.A	F.A F.A.D

LEVEL 2
COMPONENTS OF STATE T
SUBTASK STEP CONTROLLED ABORT

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

T.A SUBTASK STEP CLEAN-UP

LOOKING AT THE NATURE OF THE SUBTASK STEP, SEE
WHAT LE ARE AFFECTED AND MODIFY THE STL AND CL IN THE
PROPER WAY.

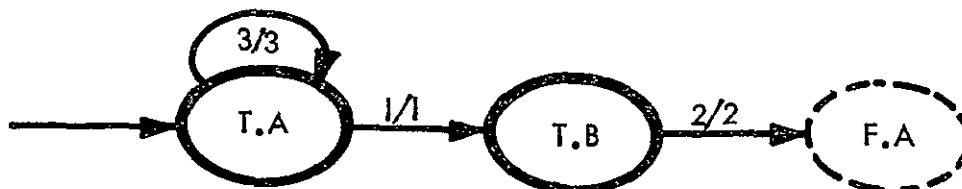
T.B ELIMINATE THE STACK ENTRY

PURGE THE SUBTASK ROLLOUT FILE AND ELIMINATE THE
TOP ENTRY IN THE PUSH DOWN STACK

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P.T.A	T.A	3	3
	T.B	1	1
T.B	P.F.A	2	2

TRANSITION DIAGRAM



T : SUBTASK STEP CONTROLLED ABORT
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	PUSH DOWN STACK
2	PURGE RESPONSE FROM OS
3	AMBIGUOUS STATUS IN AN LE (THE IMPLICATIONS OF SAVING OR PURGING THE LE ARE NOT WELL DEFINED)

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	-
3	EXPLANATION OF THE STATUS AND A REQUEST FOR A DECISION TO SAVE OR PURGE THE LE

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
T.A	F.A F.A.B

LEVEL 2
COMPONENTS OF STATE U
SUBTASK INTERRUPTION

ORIGINAL PAGE 15
POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

U.A DETERMINE EXIT MODE

DECIDE IF THE USER IS JUST INTERRUPTING OR HE ALSO
WANTS TO CONTINUE EXECUTING AFTER SIGNING OFF THE SUB-
TASK.

U.B PREPARE THE SUBTASK LE FOR THE CL

CLEAN UP FROM THE CURRENT STATE SO THAT RECOVERY
IS POSSIBLE AT A LATER TIME

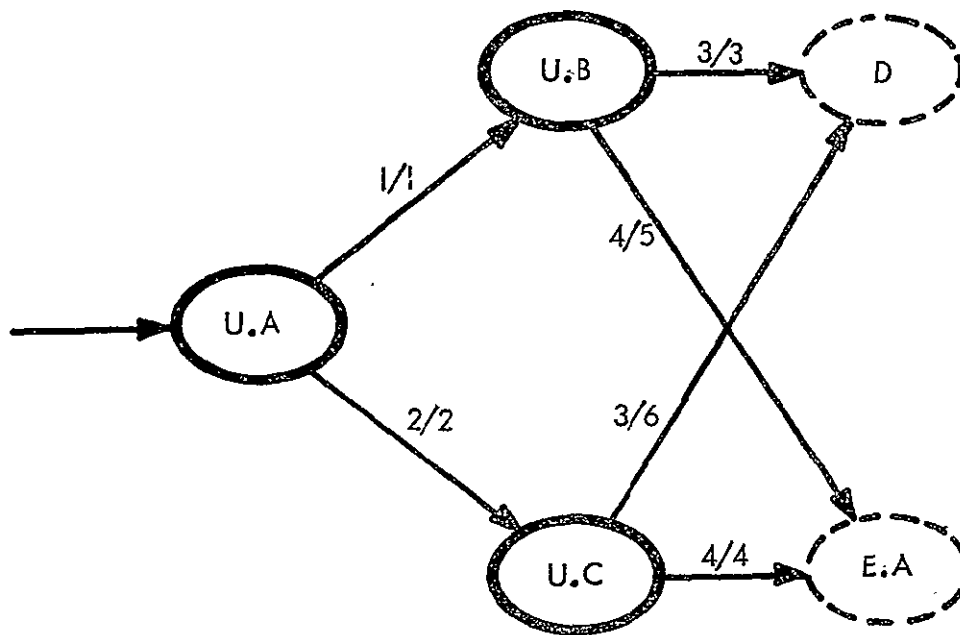
U.C PREPARE FOR EXECUTION AFTER LOG-OFF

SET UP A MACRO PROCEDURE TO EXECUTE AFTER THE
USER HAS DISCONNECTED FROM THIS SUBTASK.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P U.A	U.B	1	1
	U.C	2	2
U.B	P D	3	3
	P E.A	4	5
U.C	P D	3	6
	P E.A	4	4

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 2 TRANSITION DIAGRAM

STATE U : SUBTASK INTERRUPTION

U : SUBTASK INTERRUPTION
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	INSTRUCTIONS TO INTERRUPT IN THE CURRENT STATE
2	INSTRUCTIONS TO INITIATE EXECUTION OF A JOB OR CURRENTLY INTERRUPTED SUBTASK STEP AFTER TERMINAL SIGN
3	DONE
4	ANOTHER

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	EXECUTE COMMAND, OR A RETURN
3	COMPLETED SUBTASK LE IN THE CL, EXIT COMMAND TO OS
4	MACRO PROCEDURE TO BE EXECUTED, COMMAND TO OS TO EXECUTE THE MACRO PROCEDURE PROCESSOR, COMMAND TO THE OS TO EXECUTE THE SUBTASK SET-UP PROCESSOR.
5	COMPLETED SUBTASK IN THE CL; EXECUTION COMMAND TO THE OS TO EXECUTE THE SUBTASK SET UP PROCESSOR.
6	MACRO PROCEDURE TO BE EXECUTED, COMMAND TO THE OS TO EXECUTE THE MACRO PROCEDURE PROCESSOR, AND AN EXIT COMMAND TO THE OS.

***** CROSS REFERENCE D TRANSITIONS *****

STATE IS ACCESSIBLE FROM

U.A	F.A
	F.A.B

1

✱ ✱ ✱ ✱ ✱

T

4

4

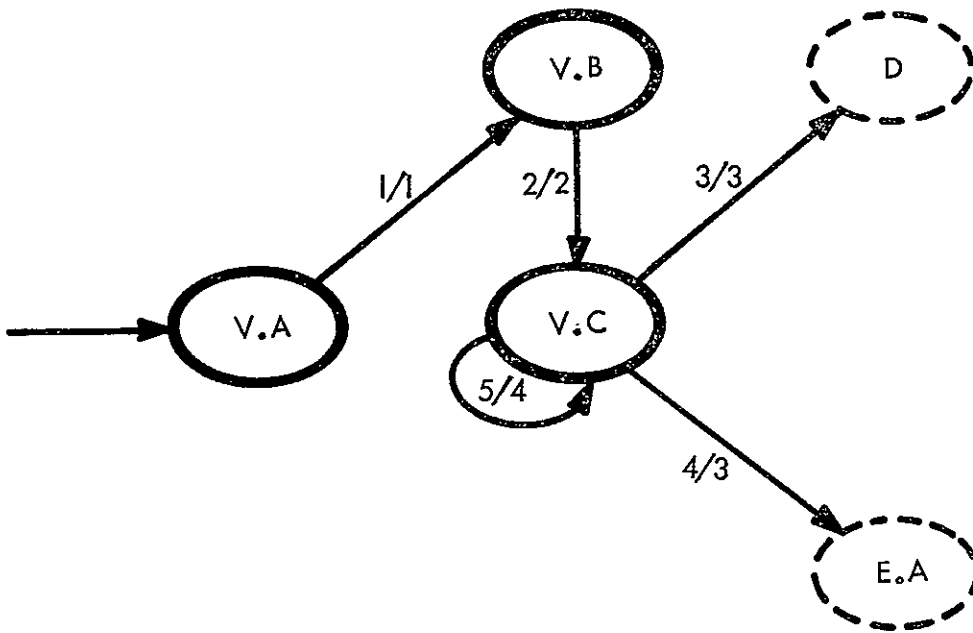
24

F

8 f

1

27



LEVEL 2 TRANSITION DIAGRAM

STATE V : SUBTASK TERMINATION

V.1 SUBTASK TERMINATION
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	ALL INFORMATION REQUIRED TO COMPLETE THE REPORTS
2	SUBTASK ACTIVITY AND ACCOMPLISHMENT RECORDS
3	JONE
4	ANOTHER
5	USER INDICATION THAT SOME MANUAL DISPOSITION IS NEEDED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	ALL REPORTS SPECIFIED IN THE PLANS AND/OR ANY OTHERS DICTATED BY THE USER.
2	ALL SUMMARY AND ACCOUNTING INFORMATION TABULATED IN THE PROJECT PLANS.
3	ALL STL RESIDENT ITEMS DISPOSED OF.
4	REQUEST FOR ADDITIONAL DISPOSITION COMMANDS

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
-------	--------------------

V.A	E.B
	E.B.B
	F.A
	F.A.O

LEVEL 2
COMPONENTS OF STATE W
DEFINING LIBRARY ENTRIES OR VARIABLES

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

W.A INTERPRET COMMAND

THE COMMAND IS ANALYZED TO DETERMINE WHETHER THE USER INTENDS TO INPUT AN ENTRY FOR A DICTIONARY IN HIS STL OR IN THE CL. THE SPECIFIC DICTIONARY AND LIBRARY ENTRY TYPE TO BE DEFINED ARE ALSO DETERMINED.

W.B VALIDATE USER

THE USER MUST HAVE PERMISSION AND ACCESS CODES THAT WILL ALLOW HIM TO CARRY OUT HIS DESIRED ACTIONS.

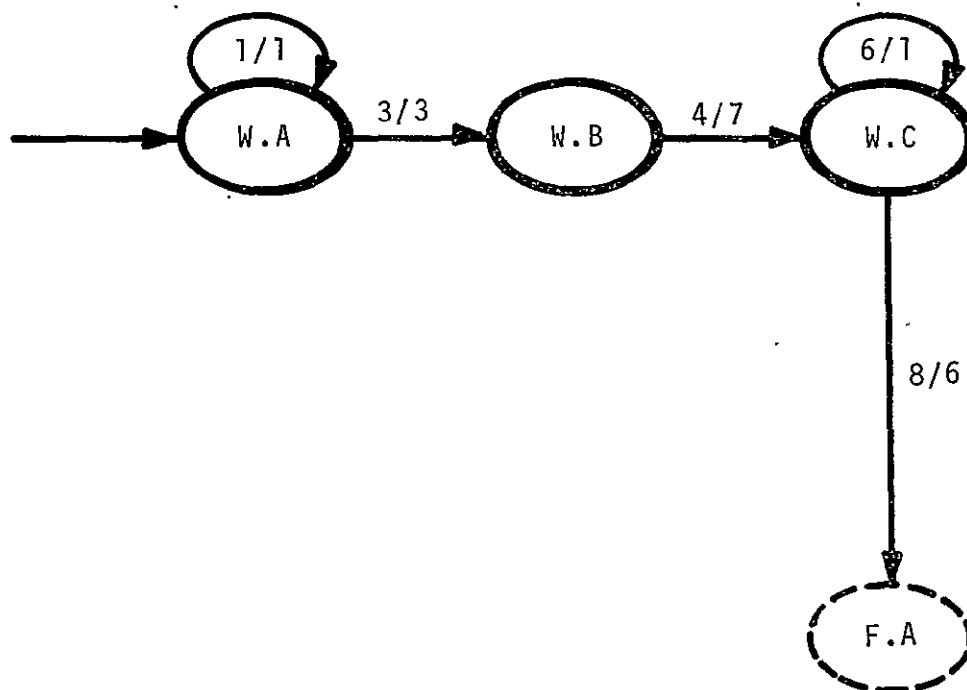
W.C CONSTRUCT DICTIONARY ENTRY

DATA FOR THE DICTIONARY ENTRY IS PROVIDED BY THE USER.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
W.A	W.A	1	1
	W.B	3	3
W.B	W.B	5	4
	W.C	4	7
W.C	W.A	3	6
	W.C	5	1

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 2 TRANSITION DIAGRAM

STATE W: DEFINING LIBRARY ENTRY OR VARIABLE

W : DEFINING LIBRARY ENTRIES OR VARIABLES
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	MORE INFORMATION REQUIRED TO COMPLETE COMMAND ANALYSIS
3	COMMAND ANALYSIS COMPLETE
4	USER VALIDATED FOR REQUESTED ACTIVITY
5	USER NOT PERMITTED TO CARRY OUT THE REQUESTED ACTION
6	MORE INFORMATION REQUIRED TO COMPLETE DICTIONARY ENTRY
3	DICTIONARY ENTRY COMPLETE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE REQUESTING USER TO ENTER MORE DATA
3	PARSED COMMAND AND COMMAND CONTROL TABLE
4	MESSAGE INDICATING REQUESTED ACTION IS NOT VALID. ASK FOR ALTERNATE COMMAND
5	DICTIONARY ENTRY AVAILABLE IN THE DATA BASE
7	MESSAGE INFORMING USER TO PROCEED

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.A	F.A F.A.D

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 3
COMPONENTS OF STATE E.A
IPAD LOG-ON

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

E.A.A USER VERIFICATION

GIVEN A USER ID AND PASSWORD, VERIFY THAT
THIS IS A VALID USER. ADDITIONAL CHECKS BEYOND THE
ID AND PASSWORD MAY HAVE TO BE MADE.

E.A.B SUBTASK VERIFICATION

GIVEN A SUBTASK IDENTIFIER, CHECK TO SEE THAT
THE SUBTASK EXISTS AS A DIRECTORY TYPE LE IN THE CL
OR AS A DEFINED ITEM IN THE PROJECT PLANS. IF THE SUB-
TASK EXISTS IN THE CL, IT MUST NOT BE ACTIVE WITH
ANOTHER USER.

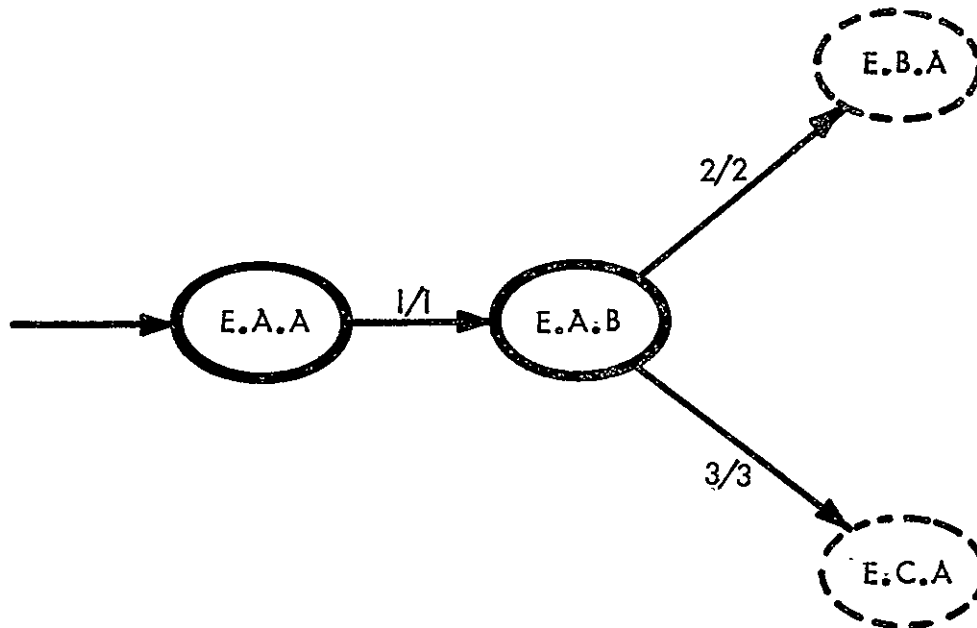
***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→E.A.A	E.A.B	1	1
E.A.B	→E.B.A	2	2
	→E.C.A	3	3

ORIGINAL PAGE IS
OF POOR QUALITY

E.A : IPAD LOG-ON
(CONTINUED)

TRANSITION DIAGRAM



***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	VALID USER NUMBER-ACCOUNT NUMBER PAIR.
2	SUBTASK IDENTIFIER= STN(PN) WHICH EXISTS IN THE CL AND IS NOT OCCUPIED BY ANOTHER USER. (STN= SUBTASK NAME, PN=PROJECT NAME)
3	SUBTASK IDENTIFIER= STN(PN) EXISTING ONLY IN THE CL LE TYPE PLAN FOR PROJECT PN.

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	LOCATION OF THE DIRECTORY ENTRY FOR THE SUBTASK
3	LOCATION OF THE DIRECTORY ENTRY FOR THE PROJECT PLAN AND THE SUBTASK NAME

LEVEL 3
COMPONENTS OF STATE E.B
RE-ACTIVATE OLD SUBTASK

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

F.3.A CHECK CURRENT STATUS

THE SUBTASK MAY OR MAY NOT HAVE AN ACTIVE STEP AT THE TIME OF SIGN ON. IF NOT, THE NEXT STATE WILL ALWAYS BE COMMAND MODE(F). IF A STEP IS ACTIVE, THE CONNECTION FROM THIS USER TO THE ACTIVE STEP MUST BE MADE.

E.3.B CONNECT TO ACTIVE STEP

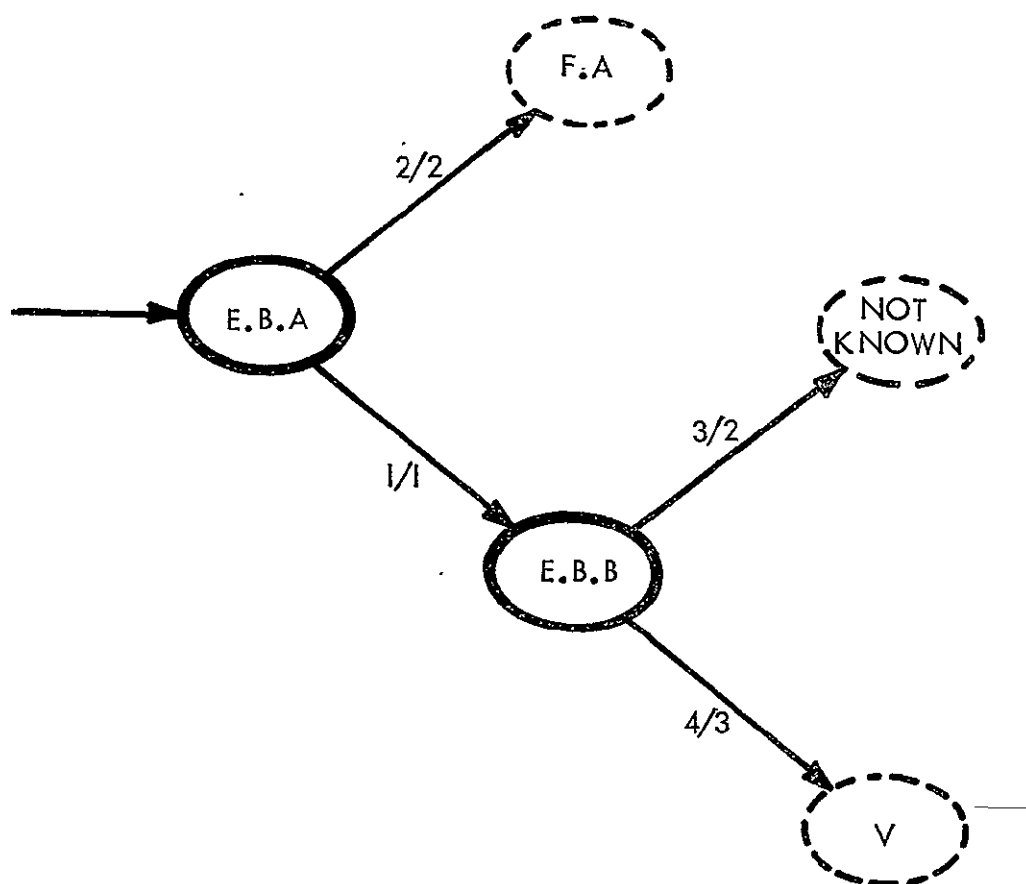
ESTABLISH THE CORRESPONDENCE BETWEEN THE ACTIVE STEP AND THIS ACTIVE USER SO THAT IT WILL BE OF NO CONSEQUENCE THAT HE INTERRUPTED.

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→E.B.A	E.B.B	1	1
	→F.A.A	2	2
E.B.B	→*	3	2
	→V**	4	3

*The state returned to is whatever state the currently executing STS represents.

**Since V was interrupted, the precise state within V cannot be specified.



LEVEL 3 TRANSITION DIAGRAM

STATE E.B : RE-ACTIVATE OLD SUBTASK

E.B : RE-ACTIVATE OLD SUBTASK
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SUBTASK DIRECTORY INDICATING THAT A STS IS CURRENTLY EXECUTING
2	INACTIVE SUBTASK DIRECTORY IN CL, WHICH WAS NOT INTERRUPTED DURING STATE V
3	CONNECT SUCCESSFUL MESSAGE FROM THE OS
4	INACTIVE SUBTASK DIRECTORY IN THE CL WHICH WAS INTERRUPTED DURING STATE V

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	LAST TERMINAL OUTPUT MESSAGE
3	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
E.B.A	E.A.B

LEVEL 3
COMPONENTS OF STATE E.C
CREATE NEW SUBTASK

ORIGINAL PAGE 1
POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

E.C.A SET UP SUBTASK DIRECTORY IN THE CL

CREATE A LE IN THE CL OF TYPE DIRECTORY WITH THE
NAME OF STN(PN) WHERE STN=SUBTASK NAME AND PN=PROJECT
NAME.

E.C.B SET UP SUBTASK RECORDS LE IN THE STL

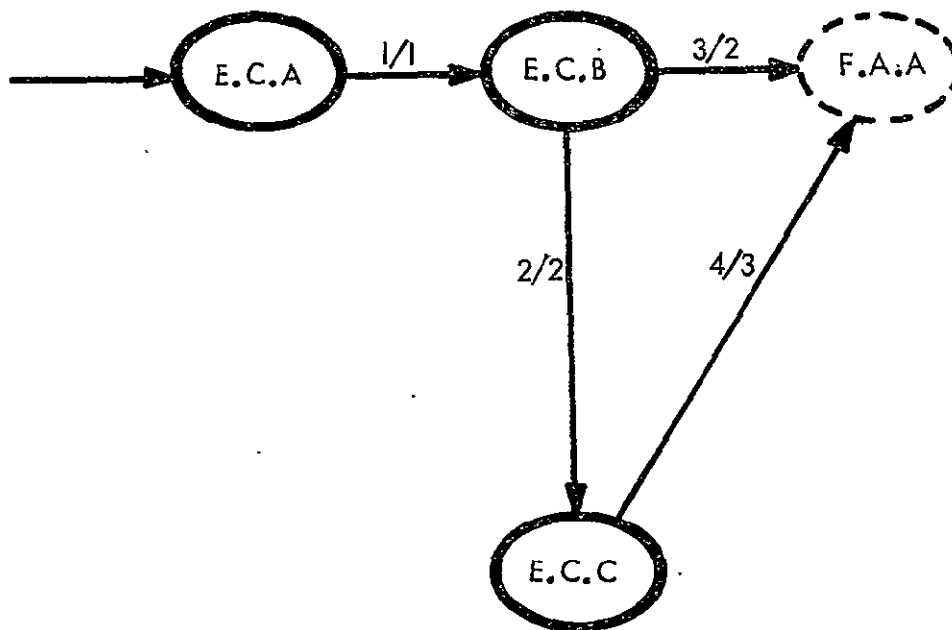
USING INFORMATION FROM THE PROJECT PLANS, THE
ACCESS AND PERMISSION CODE TABLES WILL BE FORMULATED.
THE ACTIVITY RECORD WILL BE INITIATED, ALONG WITH THE
ACCOUNTING RECORD.

E.C.C LIBRARY ENTRY INITIATION

INITIALIZE ANY LE THAT ARE KNOWN TO BE ASSOCIATED
WITH THE SUBTASK PER THE PROJECT PLANS. THIS SHOULD
ALWAYS INCLUDE A REPORT SKELETON.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→E.C.A	E.C.B	1	1
E.C.B	E.C.C	2	2
	→F.A.A	3	2
E.C.C	→F.A.A	4	3



LEVEL 3 TRANSITION DIAGRAM

STATE E.C : CREATE NEW SUBTASK

E.C : CREATE NEW SUBTASK
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	VALID SUBTASK NAME NOT CURRENTLY IN THE CL. POINTER TO ASSOCIATED PROJECT PLAN.
2	SUBTASK RECORD FROM THE PROJECT PLAN CONTAINING AT LEAST 1 LE TO BE INITIALIZED
3	SUBTASK RECORD FROM THE PROJECT PLAN WITH NO LE TO BE INITIALIZED.
4	LE SPECIFICATIONS FOR INITIAL SUBTASK LIBRARY ENTRIES

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	NEW DIRECTORY ENTRY IN THE CL FOR THIS SUBTASK
2	SUBTASK RECORDS LE INITIALIZED IN THE STL
3	LE SET UP IN STL

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
E.C.A	E.A.B

LEVEL 3
COMPONENTS OF STATE F.A.
REQUEST USER INPUT AND INTERPRET COMMAND

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

F.A.A ASK FOR, READ, AND PARSE USERS COMMAND

AFTER PROMPTING THE USER TO INSERT AN
IPAD COMMAND, THE COMMAND IS READ AND
THEN THE CHARACTER STRING FOR THE USER COMMAND IS
PARSED TO PRODUCE CONSTITUENT PARTS, THE MOST IMPORTANT
BEING THE COMMAND VERB

F.A.B DETERMINE COMMAND INTENT

THE COMMAND INTENT MAY OR MAY NOT BE LEGITIMATE
AND IT MUST BE CHECKED AGAINST THE CURRENT LIST.

F.A.C VERIFY PERMISSION TO USE COMMAND

USING THE SUBTASK RECORDS, CHECK TO SEE THAT THIS
USER MAY EXECUTE THIS COMMAND.

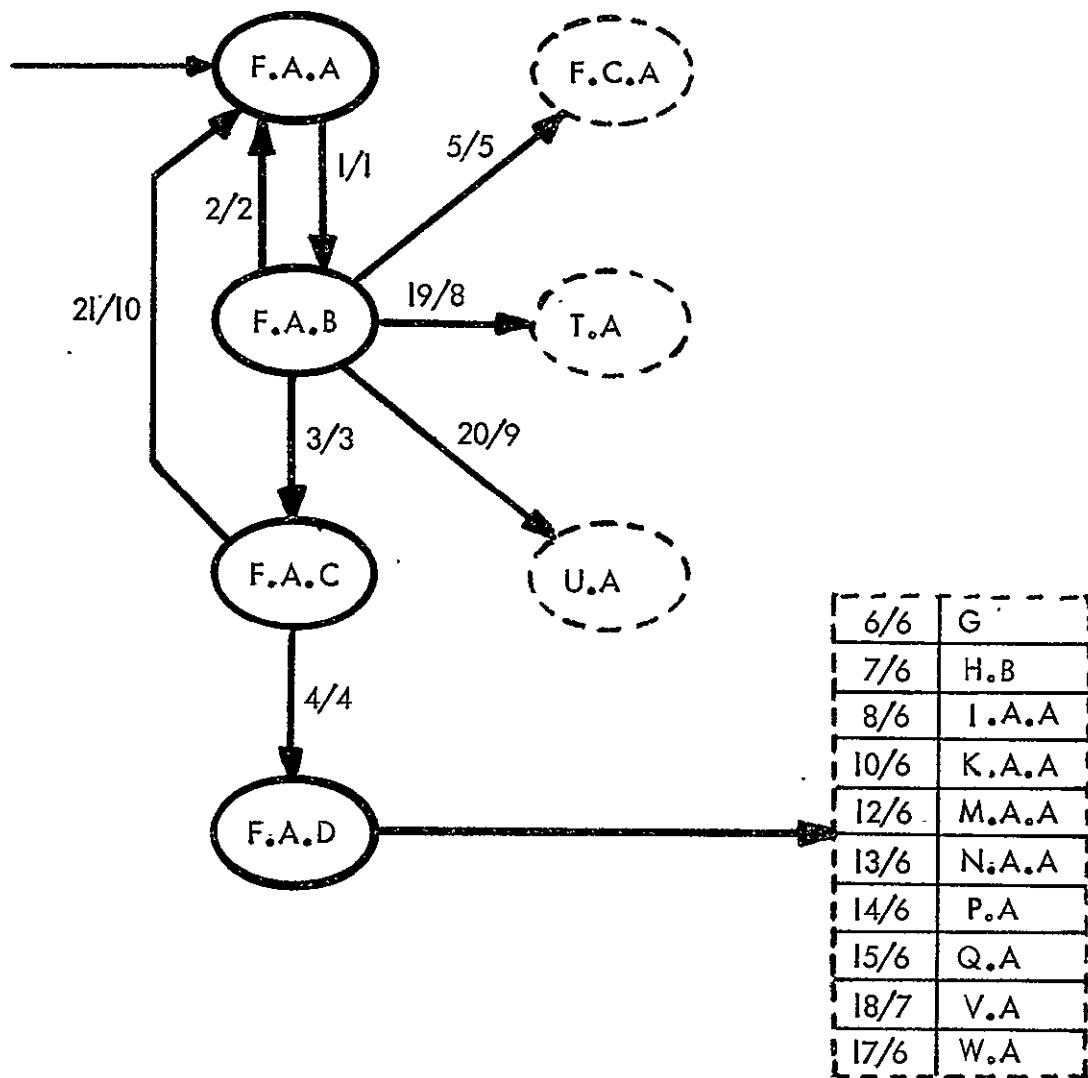
F.A.D ACTIVATE IPAD UTILITY.

INITIATE THE ACTIVITY REQUESTED FOR IN THE
COMMAND.

F.A : REQUEST USER INPUT AND INTERPRET COMMAND
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P F.A.A	F.A.B	1	1
F.A.B	F.A.A	2	2
	F.A.C	3	3
	P F.C.A	5	5
	P T.A	19	6
	P U.A	25	5
F.A.C	F.A.A	21	10
	F.A.D	4	4
F.A.D	P G	6	6
	P H.B	7	6
	P I	8	6
	P K.A.A	10	6
	P M.A.A	12	6
	P N.A.A	13	6
	P P.A	14	6
	P Q.A	15	6
	P V.A	16	7
	P W.A	17	6



LEVEL 3 TRANSITION DIAGRAM

STATE F.A : INTERPRET COMMAND

F.A : REQUEST USER INPUT AND INTERPRET COMMAND
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	CHARACTER STRING FORMATTED CORRECTLY FOR AN IPAD COMMAND.
2	UNRECOGNIZABLE COMMAND VERB
3	VALID IPAD COMMAND VERB
4	PERMISSION CODE INDICATING USE IS VALID
5	RETURN
6	HELP
7	SEARCH
8	ENTER DATA
10	MODIFY DATA
12	CONSTRUCT JOB
13	EXECUTE
14	DISPLAY
15	DISPOSE
17	DEFINE
18	TERMINATE
19	STOP
20	QUIT
21	PERMISSION CODE INDICATING USE IS INVALID

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	COMMAND VERB, ANY OTHER INFORMATION SUPPLIED WITH THE VERB
2	ERROR MESSAGE, REQUEST FOR ANOTHER TRY
3	-
4	-
5	PUSH DOWN STACK
6	PARSED COMMAND, UPDATED ACTIVITY RECORD
7	-
8	POINTER TO THE STS TO BE TERMINATED
9	-
10	ERROR MESSAGE, REQUEST FOR ANOTHER TRY

F.A : REQUEST USER INPUT AND INTERPRET COMMAND
(CONTINUED)

***** CROSS REFERENCED TRANSITIONS *****

STATE IS ACCESSIBLE FROM

F.A.A	E.B.A
	E.C.B
	E.C.C
	F.B.B
	H.C.B
	H.C.C
	H.D.B
	H.D.C
	H.D.D
	H.J.E
	M.C.D

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 3
COMPONENTS OF STATE F.B
DE-ACTIVATE SUBTASK STEP

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

F.B.A PREPARE SUBTASK STEP FILES

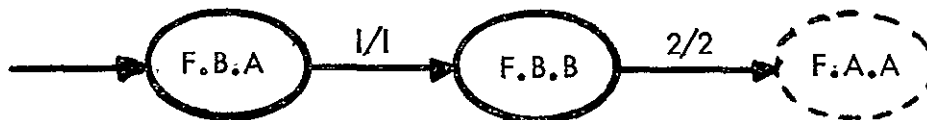
LOCATE ALL THE FILES ASSOCIATED WITH THIS STS AND
PACKAGE THEM UP FOR RECOVERY AT A LATER TIME. THIS WILL
INTERFACE WITH THE OS.

F.B.B ADJUST STACK
PUT THIS STS IN THE STACK

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→F.B.A	F.B.B	1	1
F.B.B	→F.A.A	2	2

TRANSITION DIAGRAM



F.8 : DE-ACTIVATE SUBTASK STEP
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	PUSH DOWN STACK FOR INTERRUPTED STS; RECOVERY INFORMATION FROM THE STS ROLLOUT FILE
2	PUSH DOWN STACK UPDATE INFORMATION

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	COMPLETELY DE-ACTIVATED STS PUSH DOWN STACK
2	UPDATED PUSH DOWN STACK

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 3
COMPONENTS OF STATE F.C
RE-ACTIVATE SUBTASK STEP

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
-------	--------------------

F.C.A	LOCATE STS
-------	------------

USING THE PUSH DOWN STACK, IDENTIFY THE STS TO BE
RE-ACTIVATED

F.C.B	PREPARE STS FOR RE-ACTIVATION
-------	-------------------------------

THIS IS BASICALLY THE INVERSE OF F.B.A AS IT
PREPARES ALL THE FILES OF THE STS FOR EXECUTION.

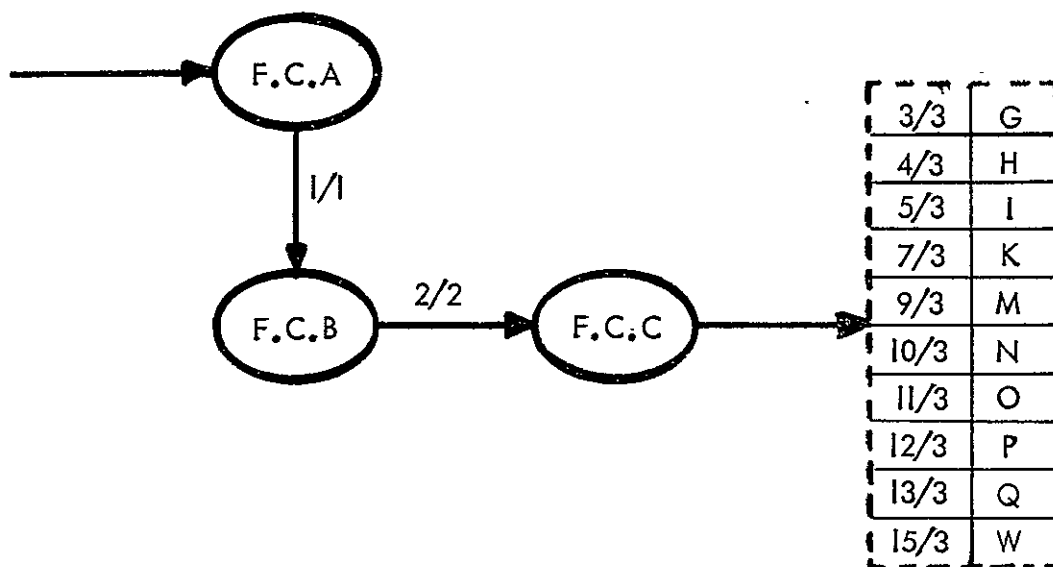
F.C.C	ACTIVATE STS
-------	--------------

RE-ISSUE THE LAST TERMINAL MESSAGE AND REQUEST THE
OS TO ACTIVATE THE STS

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→F.C.A	F.C.B	1	1
F.C.B	F.C.C	2	2
F.C.C *	→G	3	3
	→H	4	3
	→I	5	3
	→K	7	3
	→M	9	3
	→N	10	3
	→O	11	3
	→P	12	3
	→Q	13	3
	→W	15	3

*Since these are transitions to interrupted states, the
node names at level 3 cannot be specified.



LEVEL 3 TRANSITION DIAGRAM

STATE F.C : RE-ACTIVATE SUBTASK STEP

F.C : RE-ACTIVATE SUBTASK STEP
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	PUSH DOWN STACK AND THE LOCATION OF STS ROLLOUT FILE AND RECOVERY INFORMATION FOR THE STS AT THE TOP OF THE STACK
2	PROPER RESTORE CODES ON ALL STS FILES
3	STS FILE CONTAINING AN INTERRUPTED STATE G
4	STS FILE CONTAINING AN INTERRUPTED STATE H
5	STS FILE CONTAINING AN INTERRUPTED STATE I
7	STS FILE CONTAINING AN INTERRUPTED STATE K
9	STS FILE CONTAINING AN INTERRUPTED STATE M
10	STS FILE CONTAINING AN INTERRUPTED STATE N
11	STS FILE CONTAINING AN INTERRUPTED STATE O
12	STS FILE CONTAINING AN INTERRUPTED STATE P
13	STS FILE CONTAINING AN INTERRUPTED STATE Q
15	STS FILE CONTAINING AN INTERRUPTED STATE W

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	STS FILE POINTER(S)
2	ALL STS FILES READY TO EXECUTE
3	LAST RECORDED LINE SENT TO THE TERMINAL MODIFIED PUSH DOWN STACK

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
F.C.A	F.A.B

LEVEL 3
COMPONENTS OF STATE H.C
USER CONTROLLED SEARCH

STATE DESCRIPTIONS

STATE	LONG NAME AND TEXT
-------	--------------------

H.C.A DETERMINE SEARCH MODE

THE USER HAS ALREADY INDICATED THAT HE WANTS TO CONTROL THE LIBRARY SEARCH. HE ENTERS INTO ADDITIONAL DIALOG, IF NECESSARY, TO SPECIFY WHAT HE IS LOOKING FOR AND HOW HE WANTS TO INTERACT WITH THE SYSTEM

H.C.B. PERFORM SINGLE ITEM SEARCH

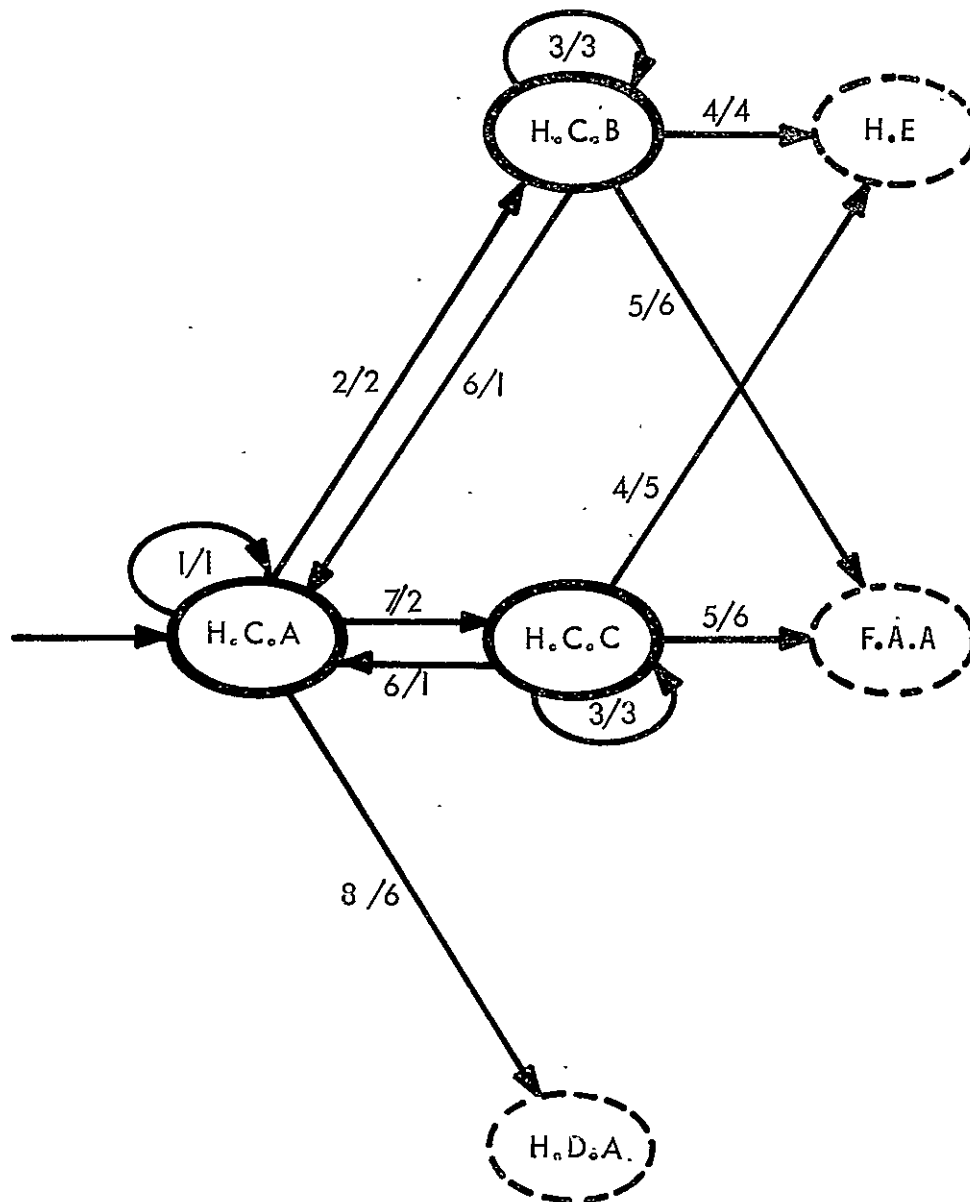
THE USER HAS REQUESTED AN EXISTENCE SEARCH FOR A SINGLE ITEM. IF FOUND HE MAY CHOOSE TO DISPLAY THE ITEM

H.C.C PERFORM PAGED SEARCH

THE USER WANTS TO PAGE THROUGH A DIRECTORY OR A DICTIONARY. WHEN EXAMINING DIRECTORY ENTRIES THE USER MAY REQUEST DISPLAYS OF INDIVIDUAL LIBRARY ENTRIES.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
H.C.A	H.C.A	1	1
	H.C.B	2	2
	H.C.C	7	2
H.C.B	H.D.A	8	6
	F.A.A	5	6
	H.C.A	6	1
	H.C.B	3	3
H.C.C	H.E	4	4
	F.A.A	5	6
	H.C.A	6	1
	H.C.C	3	3
	H.E	4	5



LEVEL 3 TRANSITION DIAGRAM

STATE H.C: USER CONTROLLED SEARCH

H.C : USER CONTROLLED SEARCH
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	USER/SYSTEM DIALOGUE INCOMPLETE
2	USER WANTS EXISTENCE SEARCH
3	SEARCH COMPLETED
4	DISPLAY DESIRED
5	USER FINISHED WITH SEARCH ACTIVITY
6	USER WANTS TO INITIATE A NEW SEARCH(WITHOUT DISPLAY, IF ITEM FOUND)
7	USER WANTS PAGED SEARCH
8	USER WANTS TO SWITCH TO SYSTEM CONTROLLED SEARCH

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	SYSTEM MESSAGE REQUESTING MORE DATA
2	SEARCH/SELECTION CRITERIA
3	USER COMMAND TO DISPLAY OR NOT, TO END, OR TO BEGIN NEW SEARCH
4	LOCATION OF ITEM TO BE DISPLAYED
5	LOCATION OF DICTIONARY OR DIRECTORY
6	-

LEVEL 3
COMPONENTS OF STATE H.D
SYSTEM CONTROLLED SEARCH

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

H.D.A EVALUATE SELECTION EXPRESSION

THE USER PROVIDES A SET OF INFORMATION USED BY THE SYSTEM TO SELECT AND EXTRACT DATA FOR DISPLAY. THE EXPRESSION EVALUATION ALSO DETERMINES THE TYPE OF SEARCH WHICH WILL BE UNDERTAKEN.

H.D.B KEYWORD SEARCH

KEYWORDS IN DICTIONARY ENTRIES (EITHER CL OR STL) ARE USED TO LOCATE LIBRARY ENTRIES.

H.D.C REFERENCE SEARCH

EXPLICIT REFERENCES SUCH AS *USED BY* ARE USED TO LOCATE LIBRARY ENTRIES.

H.D.D ATTRIBUTE SEARCH

SIMILAR TO KEYWORD SEARCH. ATTRIBUTE INDEXES CAN BE ESTABLISHED BY THE USER EXPLICITLY BY UTILIZING LE TYPE DIRECTORY. THE DATA BASE MANAGEMENT SYSTEM WILL INCLUDE FACILITIES FOR ESTABLISHMENT AND MAINTENANCE OF ATTRIBUTE INDEXES BY SUPPORTING FILE INVERSION.

H.D.E VALUE (CONTENT) SEARCH

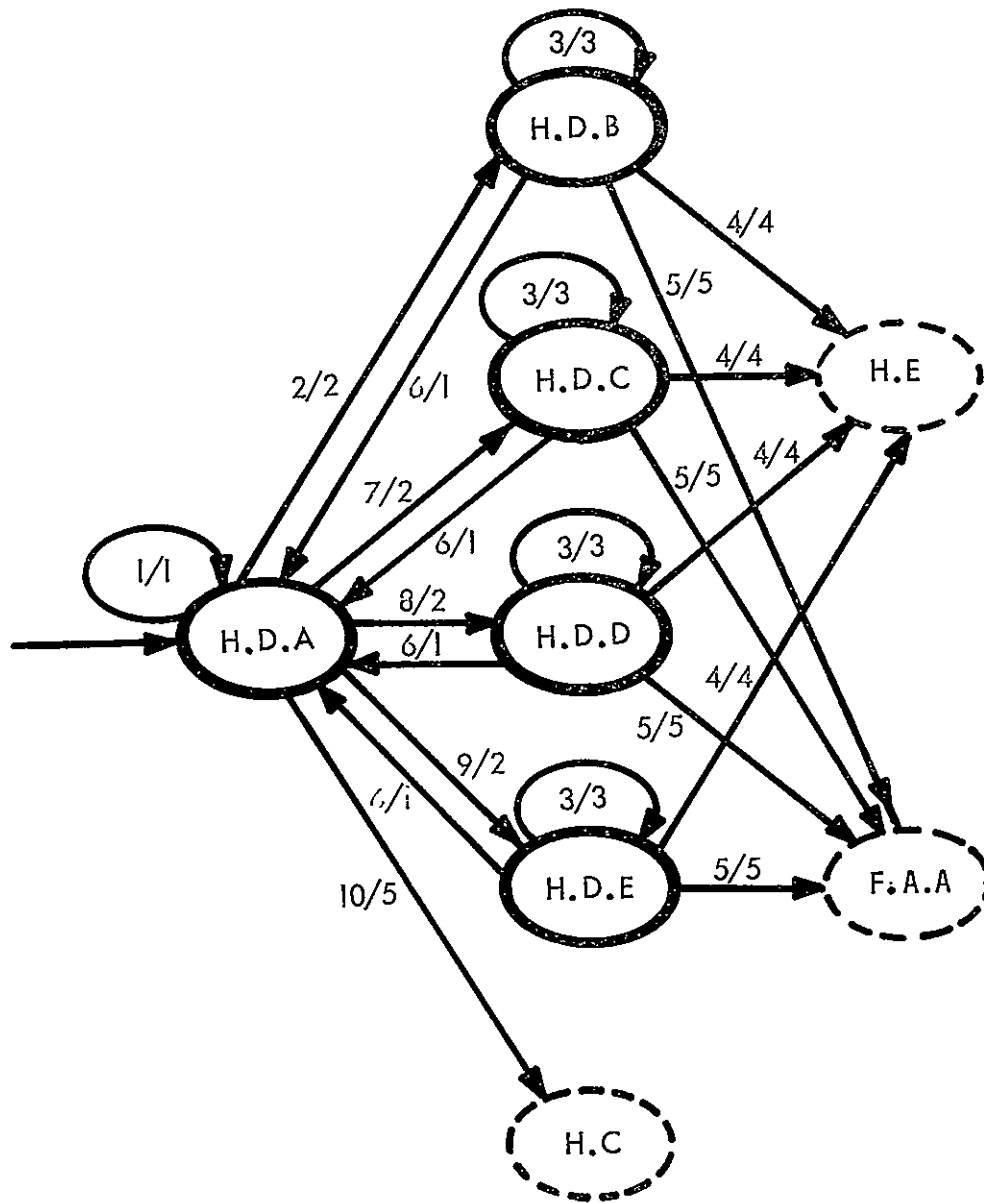
DATA AGGREGATES ARE SELECTED BASED ON VALUES OF VARIABLES. THIS FORM OF SEARCH MAY RANGE OVER MORE THAN ONE DATA SET.

H.D : SYSTEM CONTROLLED SEARCH
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→H.D.A	→H.C	13	5
	H.D.A	1	1
	H.D.B	2	2
	H.D.C	7	2
	H.D.D	8	2
	H.D.E	9	2
H.D.B	→F.A.A	5	5
	H.D.A	6	1
	H.D.C	3	3
	→H.E	4	4
H.D.C	→F.A.A	5	5
	H.D.A	6	1
	H.D.C	3	3
	→H.E	4	4
H.D.D	→F.A.A	5	5
	H.D.A	6	1
	H.D.C	3	3
	→H.E	4	4
H.D.E	→F.A.A	5	5
	H.D.A	6	1
	H.D.E	3	3
	→H.E	4	4



LEVEL 3 TRANSITION DIAGRAM

STATE H.D : SYSTEM CONTROLLED SEARCH



H.D : SYSTEM CONTROLLED SEARCH
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	USER/SYSTEM DIALOG INCOMPLETE
2	KEYWORD SEARCH REQUIRED
3	SEARCH COMPLETED
4	DISPLAY DESIRED
5	USER FINISHED WITH SEARCH ACTIVITY
6	USER WANTS TO INITIATE NEW SEARCH
7	USER WANTS REFERENCE SEARCH
8	USER WANTS ATTRIBUTE SEARCH
9	USER WANTS VALUE (CONTENT) SEARCH
10	USER WANTS TO SWITCH TO USER CONTROLLED SEARCH

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	SYSTEM MESSAGE REQUESTING MORE DATA
2	SEARCH/SELECTION CRITERIA
3	USER COMMAND TO DISPLAY OR NOT, TO END, OR TO BEGIN NEW SEARCH
4	LOCATION OF DATA TO BE DISPLAYED
5	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
H.D.A	H.C.A

LEVEL 3
COMPONENTS OF STATE I.C
CONSTRUCT LIBRARY ENTRY

ORIGINAL PAGE 1
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
I.C.A	SELECT APPROPRIATE PROCESSING MODE
I.C.B	ENTER CODING MODULE
I.C.C	ENTER DATA SET
I.C.D	ENTER STORED DATA DEFINITION
I.C.E	ENTER DICTIONARY
I.C.F	ENTER DISPLAY FORMAT
I.C.G	ENTER DISPLAY MENU
I.C.H	ENTER PLAN
I.C.I	ENTER REPORT
I.C.J	ENTER DATA CONTROL DATA

THIS IS THE STATE OF ENTERING THE INITIAL DATA TO CONTROL ACCESS TO DATA AND SYSTEM FUNCTIONS. ALL SUBSEQUENT CHANGES TO THIS CONTROL INFORMATION IS DONE VIA THE MODIFY STATE.

I.C.K ESTABLISH DIRECTORY ENTRY

A DIRECTORY ENTRY IS ESTABLISHED IN THE USERS WORK AREA

ORIGINAL PAGE IS
OF POOR QUALITY

I.C : CONSTRUCT LIBRARY ENTRY
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE LONG NAME AND TEXT

I.C.L COMPLETE DIRECTORY ENTRY

 ADDITIONAL INFORMATION IS ADDED TO THE ESTABLISHED
 DIRECTORY ENTRY

I.C.M REPORT ERROR

I.C : CONSTRUCT LIBRARY ENTRY
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** ALLOWED TRANSITIONS *****

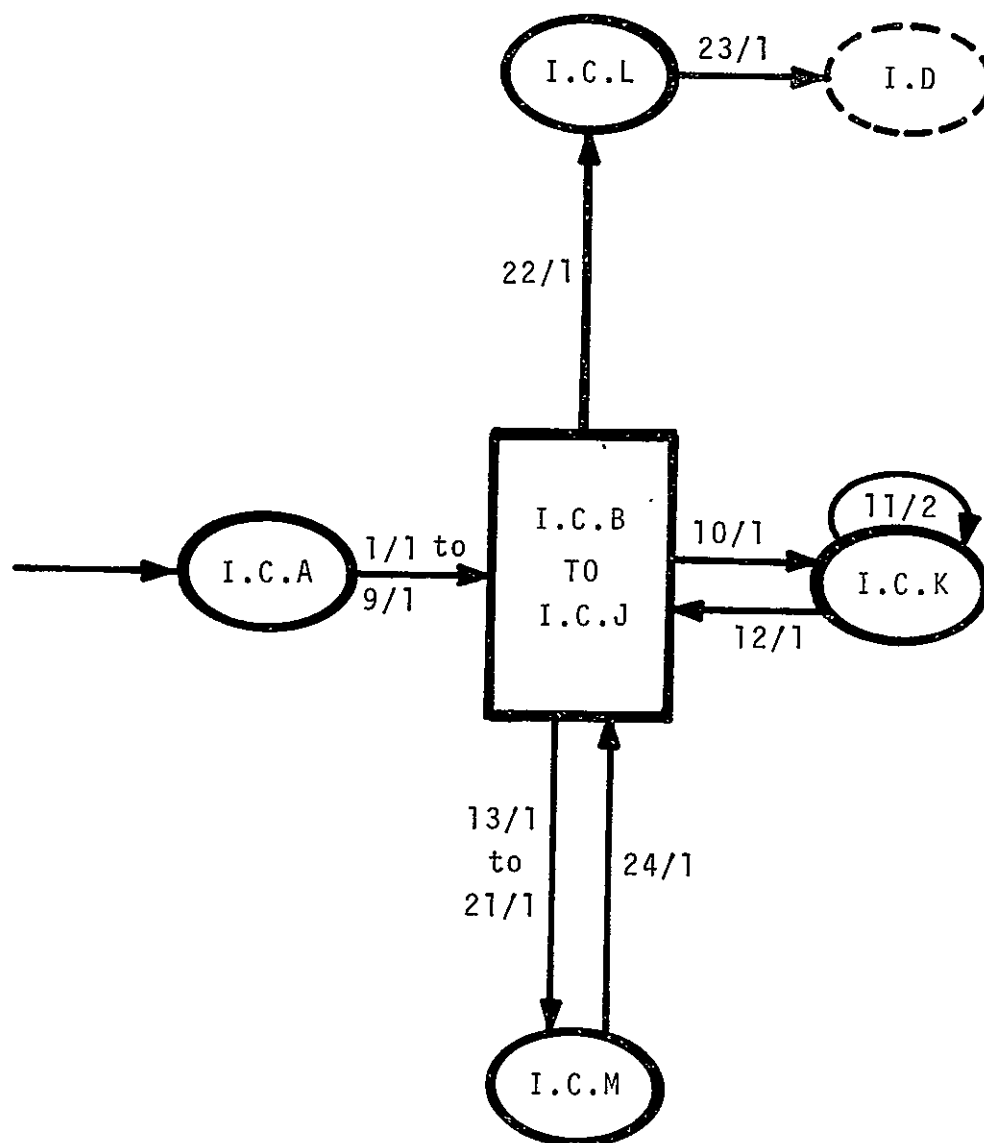
FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
I.C.A	I.C.B	1	1
	I.C.C	2	1
	I.C.D	3	1
	I.C.E	4	1
	I.C.F	5	1
	I.C.G	6	1
	I.C.H	7	1
	I.C.I	8	1
	I.C.J	9	1
	I.C.K	10	1
I.C.B	I.C.L	22	1
	I.C.M	13	1
I.C.C	I.C.K	10	1
	I.C.L	22	1
I.C.D	I.C.M	14	1
	I.C.K	10	1
I.C.E	I.C.L	22	1
	I.C.M	15	1
I.C.F	I.C.K	10	1
	I.C.L	22	1
I.C.G	I.C.M	16	1
	I.C.K	10	1
I.C.H	I.C.L	22	1
	I.C.M	17	1
I.C.I	I.C.K	10	1
	I.C.L	22	1
I.C.J	I.C.M	18	1
	I.C.K	10	1
I.C.K	I.C.L	22	1
	I.C.M	20	1
I.C.K	I.C.B	12	1
	I.C.C	12	1
	I.C.D	12	1
	I.C.E	12	1

ORIGINAL PAGE IS
OF POOR QUALITY

I.C : CONSTRUCT LIBRARY ENTRY
(CONTINUED).

***** ALLOWED TRANSITIONS (CONTINUED) *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
I.C.K	I.C.F	12	1
	I.C.G	12	1
	I.C.H	12	1
	I.C.I	12	1
	I.C.J	12	1
	I.C.K	11	2
I.C.L	I.D	23	1
I.C.M	I.C.B	24	1
	I.C.C	24	1
	I.C.D	24	1
	I.C.E	24	1
	I.C.F	24	1
	I.C.G	24	1
	I.C.H	24	1
	I.C.I	24	1
	I.C.J	24	1



LEVEL 3 TRANSITION DIAGRAM

STATE I.C: CONSTRUCT LIBRARY ENTRY

I.C : CONSTRUCT LIBRARY ENTRY
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	ENTER CM
2	ENTER OS
3	ENTER SDD
4	ENTER DIC
5	ENTER DF
6	ENTER DM
7	ENTER PLAN
8	ENTER REPORT
9	ENTER DCD
10	READY TO BEGIN LIBRARY ENTRY CONSTRUCTION
11	USER SUPPLIED DIRECTORY INFORMATION INCOMPLETE
12	DIRECTORY ENTRY ESTABLISHED IN USER WORKING SPACE.
13	ERROR MESSAGE (CONSTRUCT CM ERROR)
14	ERROR MESSAGE (CONSTRUCT OS ERROR)
15	ERROR MESSAGE (CONSTRUCT SDD ERROR)
16	ERROR MESSAGE (CONSTRUCT DIC ERROR)
17	ERROR MESSAGE (CONSTRUCT DF ERROR)
18	ERROR MESSAGE (CONSTRUCT DM ERROR)
19	ERROR MESSAGE (CONSTRUCT PLAN ERROR)
20	ERROR MESSAGE (CONSTRUCT REP ERROR)
21	ERROR MESSAGE (CONSTRUCT DCD ERROR)
22	LIBRARY ENTRY CONSTRUCTION COMPLETE, LE IN USER WORKING AREA
23	DIRECTORY ENTRY FOR NEW LE COMPLETED
24	ERROR CONDITION CLEARED, SOME RECOVERY POSSIBLE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE REQUESTING MORE INFORMATION

LEVEL 3
COMPONENTS OF STATE K.A
CONNECT USER WITH DATA TO BE MODIFIED

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

K.A.A INTERPRET COMMAND

THE MODIFY ACTIVITY POTENTIALLY INVOLVES UPDATING ANY LIBRARY ENTRY IN THE DATA BASE. THIS MAY BE ENGINEERING DATA IN A DATA SET OR SOURCE CODE IN A CM.

MODIFICATION MAY INVOLVE THE CREATION OF A NEW VERSION OF AN LE IN WHICH CASE THE PREVIOUS VERSION IS AVAILABLE UNCHANGED IN THE DATA BASE. IF MODIFICATION INVOLVES CORRECTION OF A PREVIOUS VERSION THE RETENTION OF THE PREVIOUS VERSION IS OPTIONAL.

K.A.B RETRIEVE DIRECTORY ENTRY

DIRECTORY ENTRY FOR THE LE TO BE MODIFIED IS USED FOR VALIDATION AND TO ATTACH TEXT TO USER

K.A.C VALIDATE USER

AS FOR ENTER A USER MUST HAVE PERMISSION TO CARRY OUT MODIFICATIONS. PERMISSION IS GRANTED BY PROJECT MANAGEMENT AND ADMINISTERED BY IPAD. PERMISSION MAY BE SPECIFIC WITH RESPECT TO LE TYPES, CL OR STL, SECURITY CLASSIFICATION, AND PARTICULAR OCCURRENCES

K.A.D CHECK PREVIOUS USAGE

INADVERTENT PURGING OF DATA BY REWRITING DATA THAT MIGHT STILL BE REQUIRED MUST BE AVOIDED. PREVIOUS USERS WILL BE INFORMED THAT MODIFICATIONS HAVE BEEN MADE AND THE UNMODIFIED DATA RETAINED.

K.A.E ATTACH EXISTING LE TO USER

K.A.F ATTACH COPY OF EXISTING LE TO USER

K.A : CONNECT USER WITH DATA TO BE MODIFIED
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

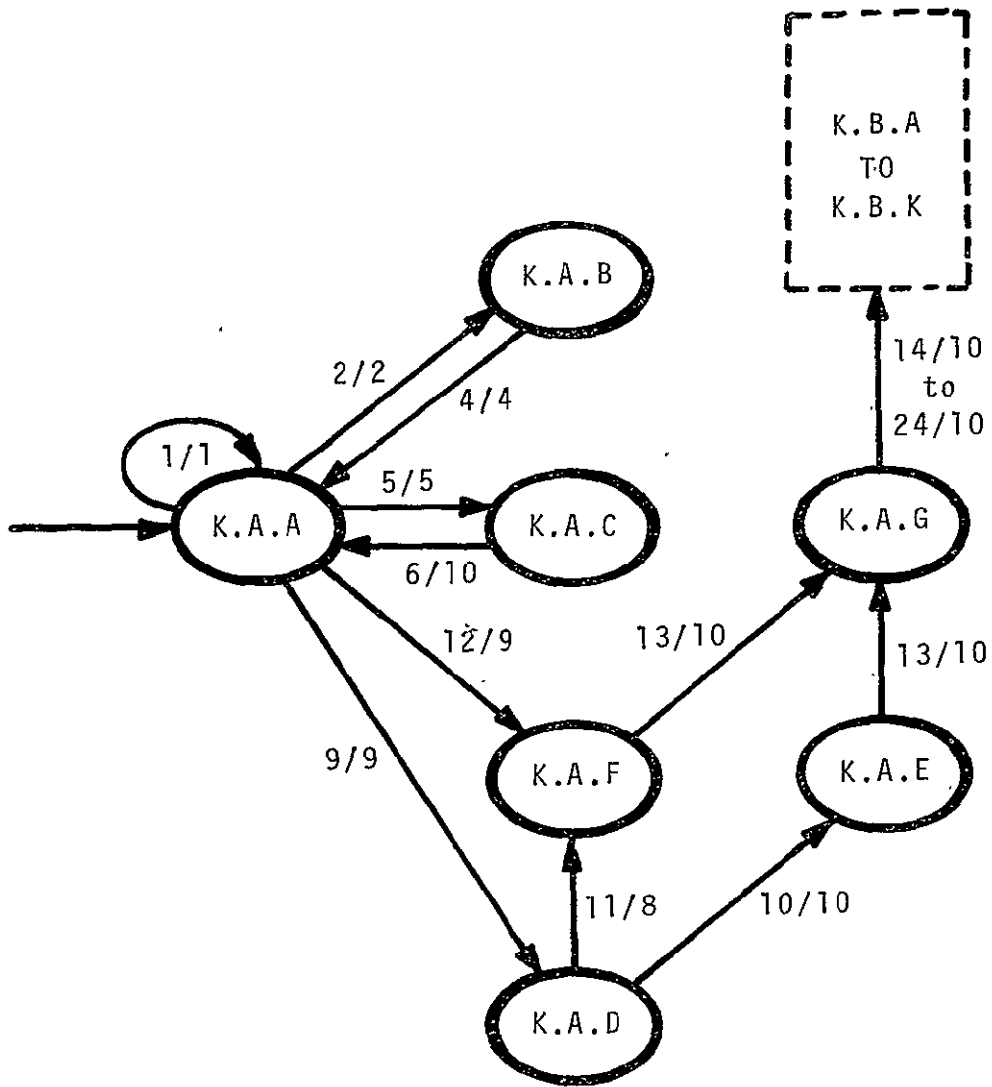
STATE LONG NAME AND TEXT

K.A.G SELECT MODIFY PROCESSOR

INFORMATION COLLECTED FROM THE COMMAND IS USED TO
DETERMINE WHICH MODIFY PROCESSOR IS TO BE USED

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.A.A	K.A.A	1	1
	K.A.B	2	2
	K.A.C	5	5
	K.A.D	9	9
	K.A.F	12	9
K.A.B	K.A.A	4	4
K.A.C	K.A.A	6	10
K.A.D	K.A.E	10	10
	K.A.F	11	5
K.A.E	K.A.G	13	10
K.A.F	K.A.G	13	10
K.A.G	→K.B.A	14	10
	→K.B.B	15	10
	→K.B.C	16	10
	→K.B.D	17	10
	→K.B.E	18	10
	→K.B.F	19	10
	→K.B.G	20	10
	→K.B.H	21	10
	→K.B.I	22	10
	→K.B.J	23	10
	→K.B.K	24	10



LEVEL 3 TRANSITION DIAGRAM

STATE K.A: CONNECT USER WITH DATA TO BE MODIFIED

K.A : CONNECT USER WITH DATA TO BE MODIFIED
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	MORE INFORMATION REQUIRED TO COMPLETE COMMAND ANALYSIS
2	LIBRARY AND LIBRARY ENTRY TO BE MODIFIED IDENTIFIED
4	DIRECTORY ENTRY FOR LE OBTAINED
5	VALIDATION REQUIRED
6	VALIDATION CHECK OK
9	VALID USER, COMMAND ANALYSIS COMPLETE, REWRITE REQUESTED
10	NO PREVIOUS USAGE OF LE TO BE MODIFIED
11	PREVIOUS LE USAGE DETERMINED
12	VALID USER, COMMAND ANALYSIS COMPLETE, USER WANTS TO PRESERVE PREVIOUS VERSION AND CREATE NEW VERSION CONSIDERED A VARIANT RATHER THAN A CORRECTION.
13	LE ENTRY ATTACHED TO USER
14	USER DESIRES TO MODIFY A CODING MODULE
15	USER DESIRES TO MODIFY A OPERATIONAL MODULE
16	USER DESIRES TO MODIFY A JOB
17	USER DESIRES TO MODIFY A DATA SET
18	USER DESIRES TO MODIFY A DISPLAY FORMAT
19	USER DESIRES TO MODIFY A DICTIONARY
20	USER DESIRES TO MODIFY A DISPLAY MENU
21	USER DESIRES TO MODIFY A PLAN
22	USER DESIRES TO MODIFY A REPORT
23	USER DESIRES TO MODIFY A STORED DATA DEFINITION
24	USER DESIRES TO MODIFY DATA CONTROL DATA

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE REQUESTING USER TO ENTER MODRE DATA
2	LIBRARY ID AND LE NAME, TYPE
4	DIRECTORY ENTRY IN USERS WORKING AREA
5	SPECIFIC ITEM/ACTION REQUIRING APPROVAL
8	LIST OF PREVIOUS USERS
9	PARSED COMMAND AND COMMAND CONTROL TABLE
10	-

***** CROSS REFERENCED TRANSITIONS *****

STATE IS ACCESSIBLE FROM

K.A.A

F.A.U

LEVEL 3
COMPONENTS OF STATE K.8
PERFORM MODIFICATIONS WITH DIALOG

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
K.3.A	MODIFY CM
K.3.B	MODIFY OM
K.3.C	MODIFY JOB
K.3.D	MODIFY DS
K.3.E	MODIFY DF
K.3.F	MODIFY DIC
K.3.G	MODIFY DM
K.3.H	MODIFY PLAN
K.3.I	MODIFY REPORT
K.3.J	MODIFY SSJ
K.3.K	MODIFY DCD

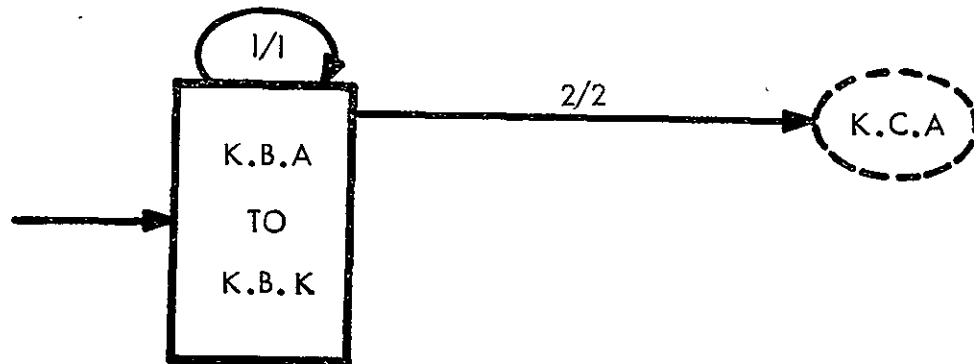
ORIGINAL PAGE IS
OF POOR QUALITY

K.B : PERFORM MODIFICATIONS WITH DIALOG
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.B.A	K.B.A	1	1
	→K.C.A	2	2
→K.B.B	K.B.B	1	1
	→K.C.A	2	2
→K.B.C	K.B.C	1	1
	→K.C.A	2	2
→K.B.D	K.B.D	1	1
	→K.C.A	2	2
→K.B.E	K.B.E	1	1
	→K.C.A	2	2
→K.B.F	K.B.F	1	1
	→K.C.A	2	2
→K.B.G	K.B.G	1	1
	→K.C.A	2	2
→K.B.H	K.B.H	1	1
	→K.C.A	2	2
→K.B.I	K.B.I	1	1
	→K.C.A	2	2
→K.B.J	K.B.J	1	1
	→K.C.A	2	2
→K.B.K	K.B.K	1	1
	→K.C.A	2	2

TRANSITION DIAGRAM



K.B : PERFORM MODIFICATIONS WITH DIALOG
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	MODIFICATIONS INCOMPLETE
2	MODIFICATIONS COMPLETE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE REQUESTING MORE DATA
2	LE TEXT COMPLETE IN USER WORKING AREA

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
K.B.A	K.A.G
K.B.B	K.A.G
K.B.C	K.A.G
K.B.D	K.A.G
K.B.E	K.A.G
K.B.F	K.A.G
K.B.G	K.A.G
K.B.H	K.A.G
K.B.I	K.A.G
K.B.J	K.A.G
K.B.K	K.A.G

LEVEL 3
COMPONENTS OF STATE K.C
UPDATE DIRECTORY ENTRY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

K.C.A UPDATE TEXT LOCATION SPECIFICATIONS(TLS)

ALL BUFFERS ARE FLUSHED MOVING ANY REMAINING DATA
OUT TO THE DATA BASE, RECORDING ADDITIONAL LOCATING IN-
FORMATION IN DIRECTORY.

K.C.B UPDATE USAGE INFORMATION

DATE OF LAST ACCESS, UID OF ACCESSER, ACCESS COUNT
ARE ENTERED IN DIRECTORY.

K.C.C UPDATE DATA SET REFERENCE TABLE

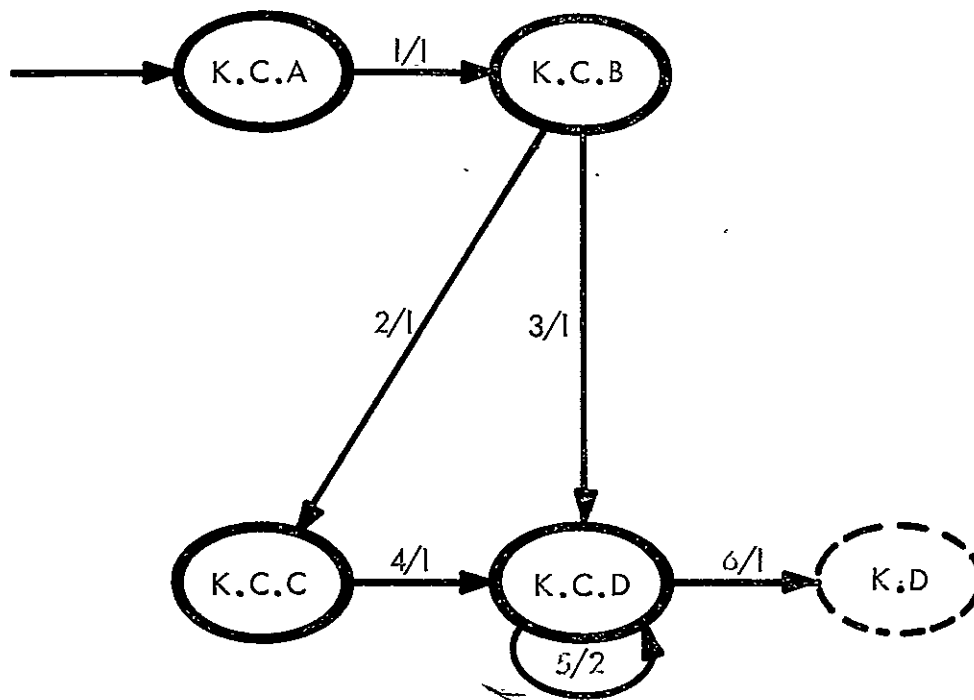
THE SUBTASK LE DATA SET REFERENCE TABLE IS UP-
DATED FOR THE DATA SET WHICH WAS MODIFIED.

K.C.D UPDATE STATUS INFORMATION

THE USER MAY CHANGE THE STATUS OF THE LE (IF HE IS
VALIDATED TO DO SO). THIS MAY INVOLVE LEVEL OF CERTI-
FICATION, ANALYSIS LEVEL, INTERNAL STRUCTURE, ETC.

***** ALLOWED TRANSITIONS *****

FROM STATE (# = ENTRY)	TO STATE (# = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.C.A	K.C.B	1	1
K.C.B	K.C.C	2	1
	K.C.D	3	1
K.C.C	K.C.D	4	1
K.C.D	K.C.D	5	2
	→K.D	6	1



LEVEL 3 TRANSITION DIAGRAM

STATE K.C : UPDATE DIRECTORY ENTRY

K.C : UPDATE DIRECTORY ENTRY
(CONTINUED).

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	BUFFERS FLUSHED, ENTIRE LIE ON DATA BASE STORAGE DEVICE
2	USAGE INFORMATION UPDATE COMPLETE. LE IS TYPE DS AND IS IN THE CL.
3	USAGE INFORMATION UPDATE COMPLETE. LE IS NOT TYPE DS.
4	DATA SET REFERENCE TABLE IN THE ST LE UPDATE COMPLETE
5	STATUS INFORMATION UPDATE NOT COMPLETE
6	STATUS INFORMATION COMPLETE

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER TO ENTER MORE INFORMATION

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
-------	--------------------

K.C.A	K.B.A
	K.B.A.B
	K.B.A.C
	K.B.B
	K.B.B.B
	K.B.B.C
	K.B.C
	K.B.C.B
	K.B.C.C
	K.B.D
	K.B.D.B
	K.B.D.C
	K.B.E
	K.B.F
	K.B.G
	K.B.H
	K.B.I
	K.B.J
	K.B.K

LEVEL 3
COMPONENTS OF STATE M.A
DETERMINE AVAILABLE JOB COMPONENTS

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

M.A.A ESTABLISH THE LIST OF OMS FOR THIS JOB
ASK FOR AND INTERPRET THE LIST OF OMS GIVEN BY
THE USER.

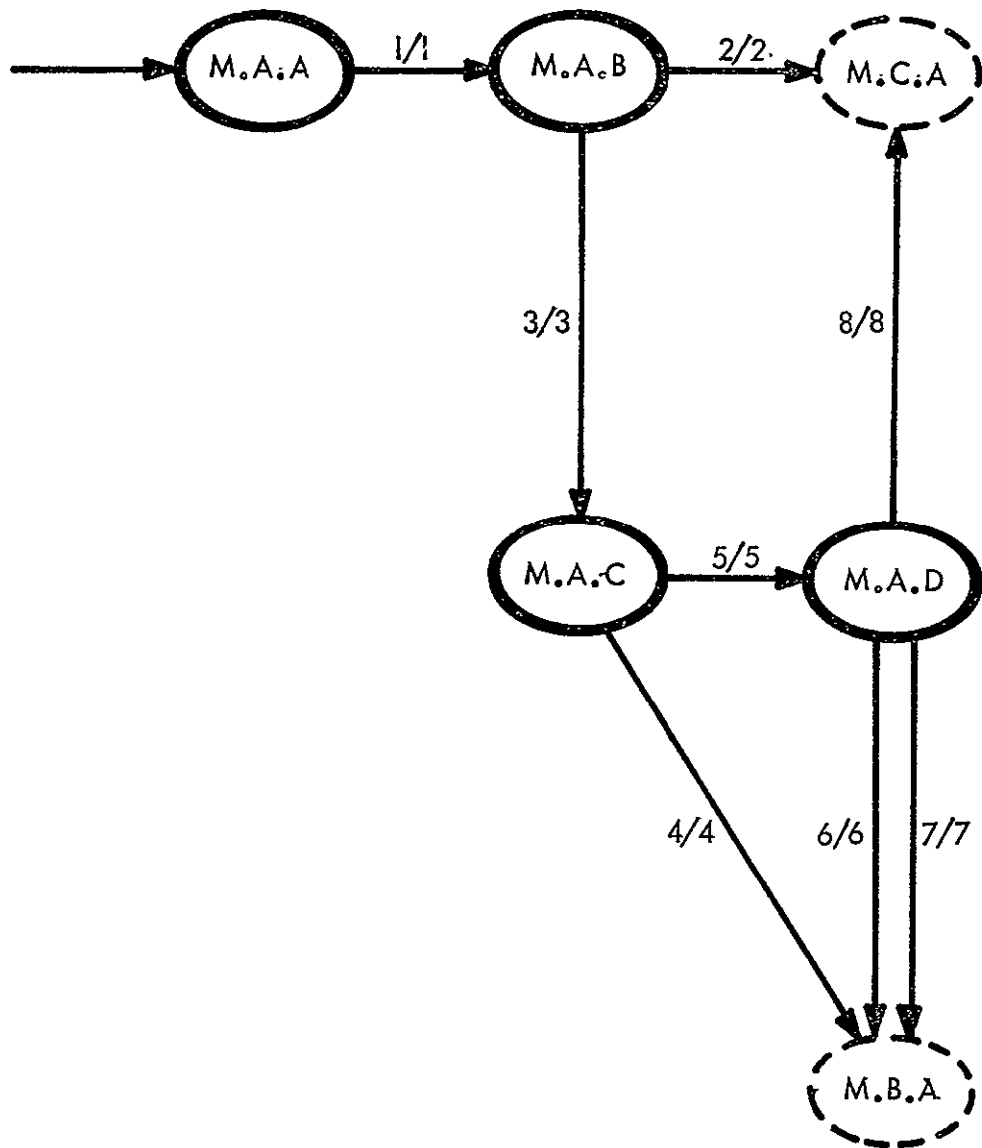
M.A.B SEARCH FOR OM NAMES IN THE STL
TRY TO SATISFY THE LIST OF REQUIRED OMS WITH THE
LIST OF OMS RESIDING IN THE STL.

M.A.C SEARCH FOR OM NAMES IN THE CL
TRY TO SATISFY THE LIST OF REQUIRED OMS WITH THE
LIST OF OMS RESIDING IN THE CL

M.A.D CHECK ACCESS TO CL RESIDENT OMS
ANY OMS REQUIRED WHICH RESIDE IN THE CL MUST BE
ACCESSABLE TO THIS USER IN EXECUTE MODE.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→M.A.A	M.A.B	1	1
M.A.B	M.A.C	3	3
	→M.C.A	2	2
M.A.C	M.A.J	5	5
	→M.B.A	4	4
M.A.D	→M.B.A	6	6
	→M.B.A	7	7
	→M.C.A	8	8



LEVEL 3 TRANSITION DIAGRAM

STATE M.A : DETERMINE AVAILABLE JOB COMPONENTS

M.A : DETERMINE AVAILABLE JOB COMPONENTS
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	LIST OF REQUIRED OMS
2	STL OM DIRECTORY CONTAINING THE NAMES OF ALL REQUIRED OMS
3	STL OM DIRECTORY CONTAINING LESS THAN ALL THE NAMES OF REQUIRED OMS
4	CL OM DIRECTORY CONTAINING NONE OF THE NAMES IN THE GIVEN SEARCH LIST
5	CL OM DIRECTORY CONTAINING AT LEAST 1 OM NAME IN THE GIVEN SEARCH LIST
6	ACCESS CODE TABLE DENYING EXECUTE PERMISSION FOR AT LEAST 1 CL RESIDENT OM
7	ACCESS CODE TABLE GIVING EXECUTE PERMISSION FOR ALL CL RESIDENT OMS FOUND, AND A CL OM DIRECTORY CONTAINING LESS THAN ALL THE REQUIRED OMS.
3	ACCESS CODE TABLE GIVING EXECUTE PERMISSION FOR ALL CL RESIDENT OMS FOUND, AND THE COMBINED STL, CL OM DIRECTORIES CONTAIN ALL REQUIRED OMS.

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	USERS OM LIST
2	LIST OF NAMES FOUND IN THE STL
3	LIST OF NAMES FOUND IN THE STL AND THOSE NOT FOUND
4	LIST OF NAMES FOUND IN THE STL AND THOSE NOT FOUND IN THE CL
5	LIST OF OMS FOUND IN THE CL AND THOSE NOT FOUND
6	LIST OF STL OMS FOUND, CL OMS FOUND AND ACCESSABLE, AND CL OMS FOUND BUT NOT ACCESSABLE
7	LIST OF FOUND AND ACCESSABLE OMS AND THOSE NOT FOUND
3	LIST OF ALL REQUIRED OMS

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
-------	--------------------

M.A.A	F.A.D
	M.B.F
	M.C.D

LEVEL 3
COMPONENTS OF STATE M.B
CONSTRUCT AN OM LIBRARY ENTRY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

M.B.A FORM INITIAL DIRECTORY ENTRY

SET UP THE DIRECTORY ENTRY PROTOTYPE AND FILL IN
CURRENTLY AVAILABLE ITEMS LIKE NAME AND TYPE.

M.B.B PROCESS FUNCTIONAL DESCRIPTION

REQUEST A FUNCTIONAL DESCRIPTION FROM THE USER,
VALIDATE THE FORM, AND INSERT IT IN THE DIRECTORY ENTRY

M.B.C PROCESS THE TEXT CONTROL DATA

REQUEST THE ELEMENTS OF THE TEXT CONTROL DATA,
VALIDATE FROM CURRENT CM LE, AND CONSTRUCT THE TCD

M.B.D CREATE THE TEXT ENTRY

GATHER ALL THE COMPONENT CMS, EXTRACT THE BINARY
DECKS, DO ANY NECESSARY PRE-LOADING, AND CONSTRUCT
THE EXECUTABLE LOAD FILES.

M.B.E CREATE CONTROL CM

IF THE CMS MAKING UP THE OM DO NOT CONTAIN A MAIN
PROGRAM, A CONTROL PROGRAM MUST BE SUPPLIED BY THE USER
AT THIS TIME.

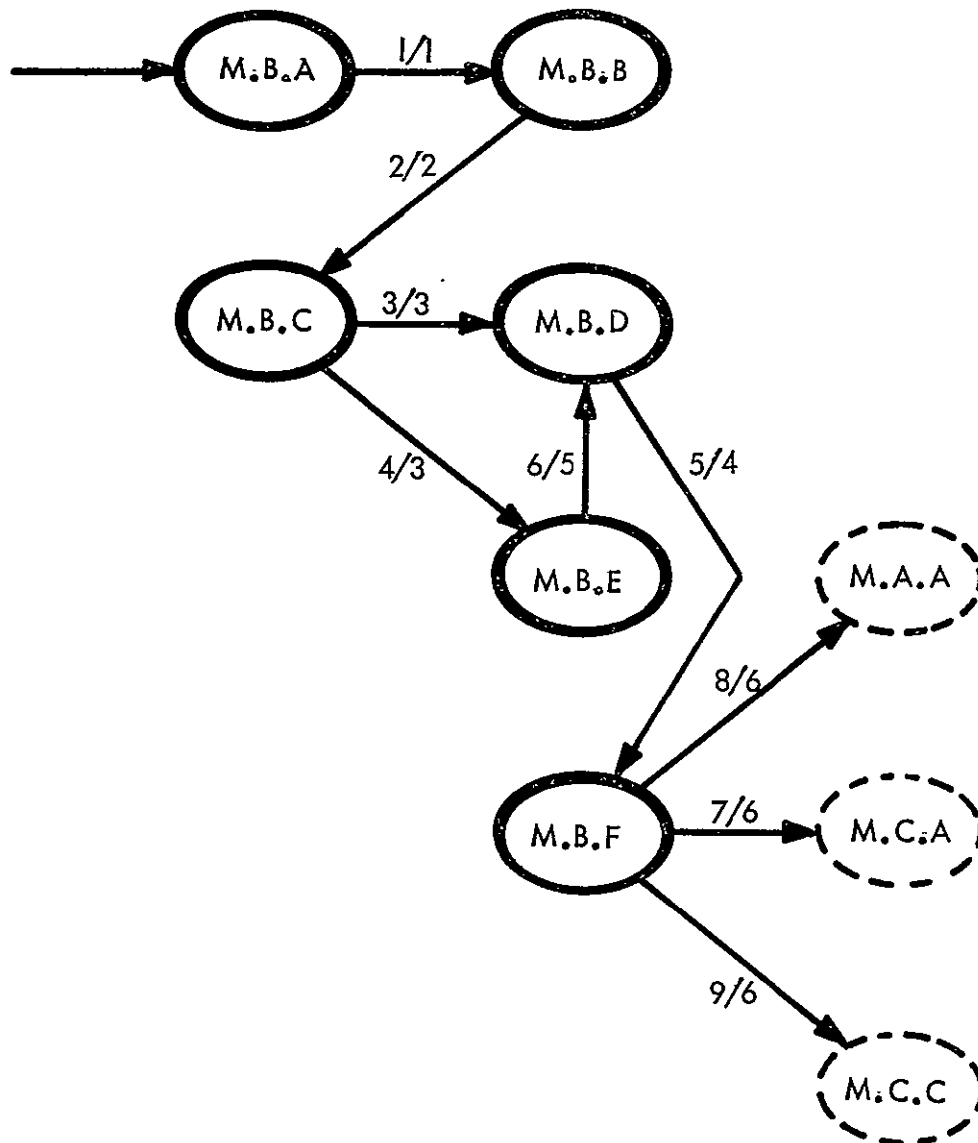
M.B.F ENTER OM INTO THE STL

MAKE THE FORMAL ENTRY OF THE OM INTO THE SUBTASK
LIBRARY.

M.B : CONSTRUCT AN OM LIBRARY ENTRY
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT.
→M.B.A	M.B.B	1	1
M.B.B	M.B.C	2	2
M.B.C	M.B.D	3	3
	M.B.E	4	3
M.B.D	M.B.F	5	4
M.B.E	M.B.D	6	5
M.B.F	→M.A.A	8	6
	→M.C.A	7	6
	→M.C.C	9	6



LEVEL 3 TRANSITION DIAGRAM

STATE M.B : CONSTRUCT AN OM LIBRARY ENTRY

M.B : CONSTRUCT AN OM LIBRARY ENTRY
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	NAME OF THE OM
2	VALID FUNCTIONAL DESCRIPTION
3	VALID USER SUPPLIED PORTION OF THE TEXT CONTROL DATA
4	LIST OF COMPONENT CMS LACKING A MAIN PROGRAM
5	DIRECTORY AND DICTIONARY INFORMATION CONSISTENT WITH THE USERS PORTION OF THE TEXT CONTROL DATA
6	VALID OM CONTROL PROGRAM
7	OM DIRECTORY INDICATING THAT ALL REQUIRED CMS ARE DEFINED AND ACCESSABLE
8	NON-EMPTY LIST OF OMS TO BE DEFINED
9	NON-EMPTY LIST OF OMS TO BE DEFINED AND AN ENTRY FLAG FROM M.C.C.

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	PARTIALLY COMPLETED DIRECTORY ENTRY
2	FUNCTIONAL DESCRIPTION PLACED IN THE DIRECTORY ENTRY
3	USERS PORTION OF THE TCD IN THE DIRECTORY ENTRY
4	COMPLETE DIRECTORY AND TEXT ENTRY FOR THE OM
5	NEW OM DEFINED IN THE SUBTASK LIBRARY, AND OUTPUT 3
6	NEW OM DEFINED IN THE SUBTASK LIBRARY

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
M.B.A	M.A.C
	M.A.D
	M.A.D
	M.C.C

LEVEL 3
COMPONENTS OF STATE M.C
CONSTRUCT A JOB LIBRARY ENTRY

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

M.C.A CONSTRUCT THE OM NETWORK

REQUEST THE NETWORK DESCRIPTION FROM THE USER AND
CONSTRUCT AN ANALYTICAL EXPRESSION OF THE NETWORK

M.C.B VALIDATE THE INTERNAL/EXTERNAL DATA FLOW

TAKING THE OM SPECIFICATIONS, CONSTRUCT THE ACTUAL
DATA FLOW WHICH WOULD OCCUR DURING EXECUTION AND ASK
FOR USER VALIDATION. NOTE THAT EXECUTION TIME DECISIONS
MAKE IT IMPOSSIBLE TO ANTICIPATE ALL POSSIBILITIES

M.C.C MODIFY THE OM NETWORK

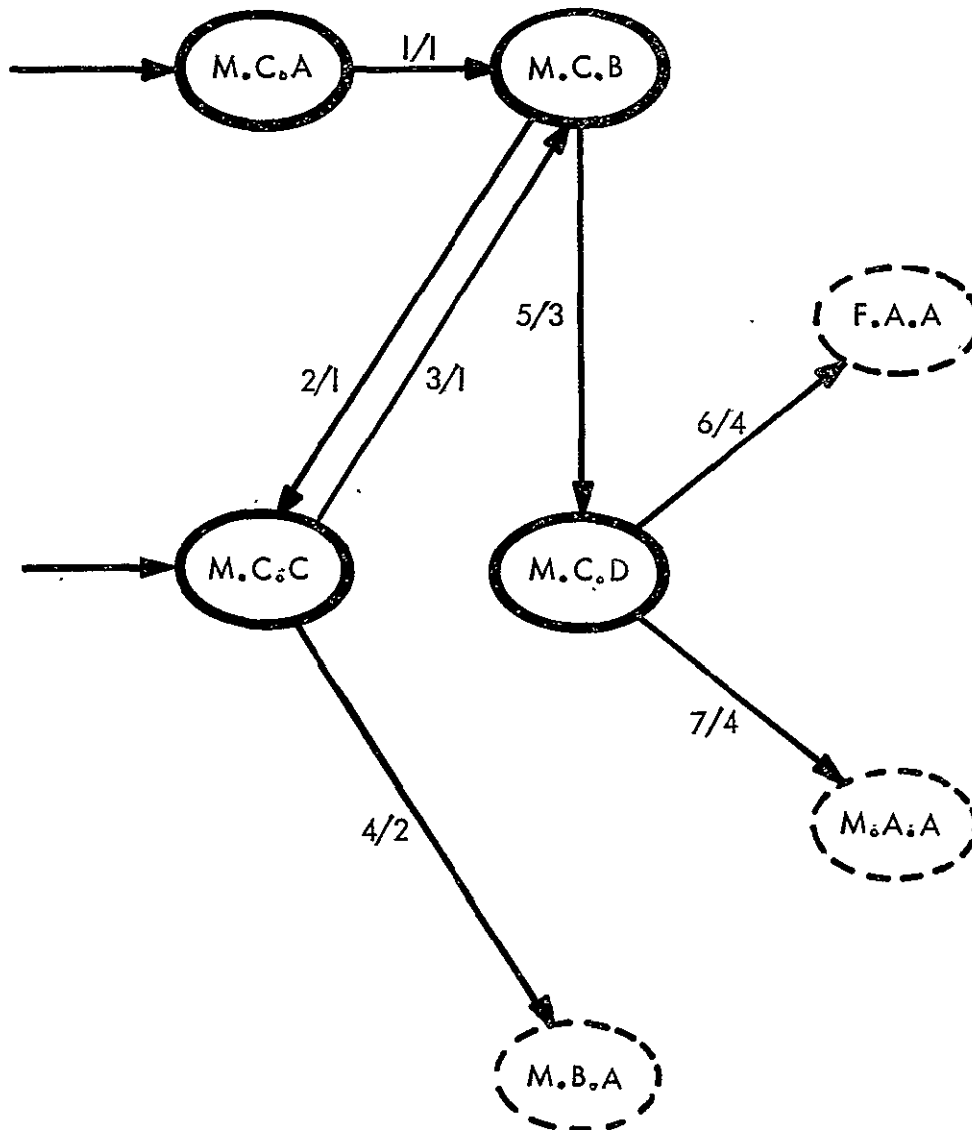
IF THE DATA FLOW IS NOT AS DESIRED, NETWORK
MODIFICATIONS MAY BE NECESSARY.

M.C.D DEFINE THE JOB IN THE SUBTASK LIBRARY

TAKING THE NETWORK DESCRIPTION AND THE COMPONENT
OMS, CONSTRUCT THE JOB DIRECTORY ENTRY AND THE SYSTEM
CONTROL CARD SKELETON RECORD.

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→M.C.A	M.C.B	1	1
M.C.B	M.C.C	2	1
	M.C.U	5	3
→M.C.C	→M.B.A	4	2
	M.C.B	3	1
M.C.D	→F.A.A	6	4
	→M.A.A	7	4



LEVEL 3 TRANSITION DIAGRAM

STATE M.C : CONSTRUCT A JOB LIBRARY ENTRY

M.C : CONSTRUCT A JOB LIBRARY ENTRY
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	VALID NETWORK SPECIFICATIONS
2	USERS RESPONSE THAT THE DATA FLOW IS NOT AS DESIRED
3	VALID NETWORK MODIFICATION SPECIFICATIONS
4	USERS RESPONSE THAT ONE OR MORE OMS ARE MISSING
5	DICTIONARY AND DIRECTORY INFORMATION COMPATIBLE WITH THE NETWORK SPECIFICATIONS.
6	EMPTY LIST OF JOBS TO BE DEFINED
7	NON-EMPTY LIST OF JOBS TO BE DEFINED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	ANALYTICAL EXPRESSION OF THE NETWORK
2	LIST OF OMS
3	ALL NECESSARY INFORMATION TO DEFINE THE JOB.
4	JOB ENTERED INTO THE SUBTASK LIBRARY

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
M.C.A	M.A.B
	M.A.D
	M.B.F
M.C.C	M.B.F

LEVEL 3
COMPONENTS OF STATE N.A
ESTABLISH THE REQUIRED LEN LIST

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
-------	--------------------

N.A.A	CONSTRUCT ULEN LIST FOR ALL I/O LE USED
	FROM THE JOB DEFINITION, FORM THE LIST OF LEN USED BY THE JOB AS INPUT, OUTPUT, OR INPUT/OUTPUT.

N.A.B	CONSTRUCT LEN FOR A DIRECTORY SEARCH
	USING THE ULEN AND THE QUALIFYING INFORMATION FROM THE EXECUTION COMMAND (EXPLICITLY GIVEN OR IMPLIED IN THE DEFAULT SENSE), CONSTRUCT THE NAMES WHICH ARE EXPECTED TO BE FOUND IN THE STL,CL SEARCH.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
N.A.A	N.A.B	1	1
N.A.B	N.B.A	2	2

TRANSITION DIAGRAM



ORIGINAL PAGE IS
OF POOR QUALITY

N.A : ESTABLISH THE REQUIRED LEN LIST
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	LIBRARY DIRECTORY ENTRY FOR THE JOB WITH EXECUTE PERMISSION FOR THIS USER.
2	QUALIFICATION INFORMATION FROM THE EXECUTE COMMAND

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	LIST OF ULEN FROM THE DIRECTORY ENTRY
2	LIST OF NAMES SUITABLE FOR A DIRECTORY SEARCH

***** CROSS REFERENCED TRANSITIONS *****

STATE IS ACCESSIBLE FROM

N.A.A

F.A.D

LEVEL 3
COMPONENTS OF STATE N.B
CHECK FOR LEN IN LIBRARIES

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

N.B.A SEARCH THE STL FOR REQUIRED LEN

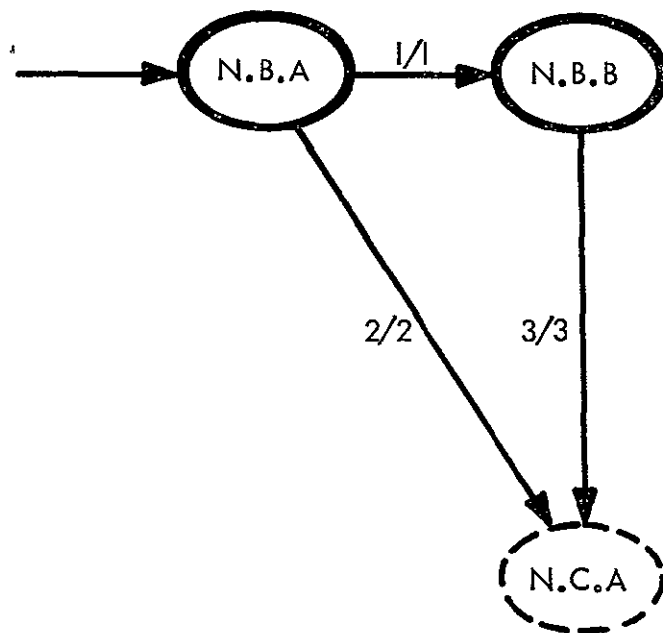
EXAMINE THE STL DIRECTORY TO FIND THE REQUIRED
LEN. NO ACCESS PERMISSION REQUIRED

N.B.B SEARCH THE CL FOR REQUIRED LEN

EXAMINE THE CL DIRECTORY TO FIND THE REQUIRED LEN.
ACCESS PERMISSION MUST BE INDICATED IN THE LED FOR THE
CORRESPONDING USE DURING EXECUTION

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
PN.B.A	N.B.B	1	1
	PN.C.A	2	2
N.B.B	PN.C.A	3	3



LEVEL 3 TRANSITION DIAGRAM

STATE N.B : CHECK FOR LEN IN LIBRARIES

N.B : CHECK FOR LEN IN LIBRARIES
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SEARCH LIST OF LEN NOT ALL CONTAINED IN THE STL
2	SEARCH LIST OF LEN ALL OF WHICH ARE CONTAINED IN THE STL
3	SEARCH LIST OF LE ALL OF WHICH ARE CONTAINED IN THE CL AND HAVE PROPER ACCESS PERMISSION CODES

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	LIST OF LEN LOCATED IN THE STL, THOSE YET TO BE LOCATED
2	LIST OF ALREADY EXISTING LIBRARY ENTRIES TO BE USED DURING EXECUTION, WITH APPROPRIATE LINKING INFORMATION.
3	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.B.A	N.A.B

LEVEL 3
COMPONENTS OF STATE N.C
PREPARE JOB FOR EXECUTION

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
-------	--------------------

N.C.A	<p>PREPARE LEN FOR ALL OUTPUT AND I/O LE</p> <p>PREPARE AHEAD OF EXECUTION ALL THE NAMES (QUALIFIED AND UNQUALIFIED) FOR THE LE TO BE CREATED DURING THE JOB. DIRECTORY ENTRIES ARE MADE WITH NO TEXT ENTRY.</p>
-------	--

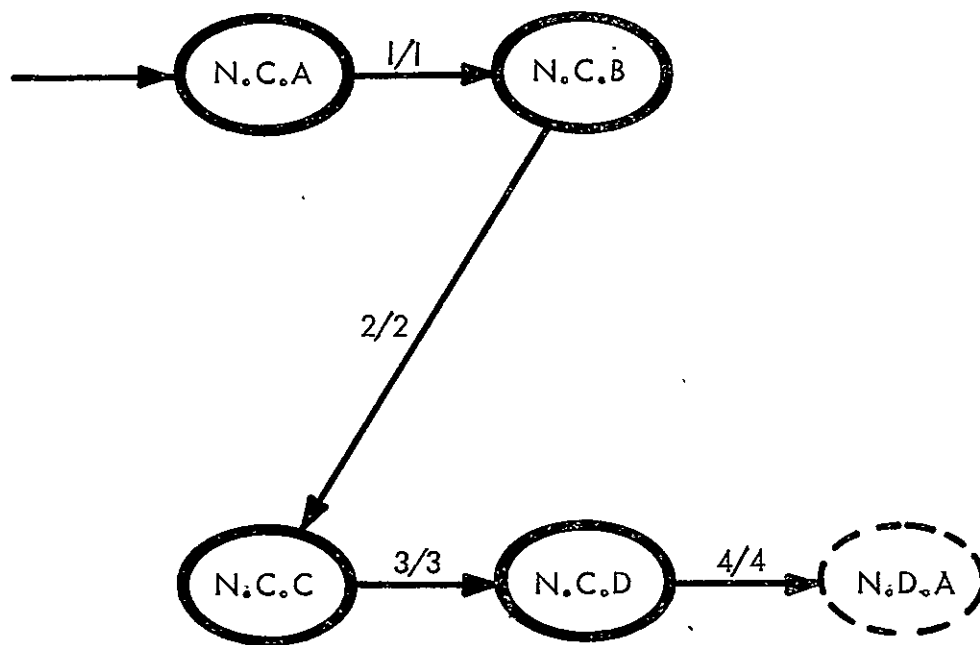
N.C.B	<p>SET UP FILE LINKAGES</p> <p>ACTUAL LOGICAL FILE NAMES USED BY THE OMS MUST BE RECONCILED WITH CURRENT LOCATIONS FOR INPUT LE AND THE ACTUAL LINKAGES TO THE LTE MUST BE SET UP.</p>
-------	--

N.C.C	<p>PREPARE THE EXECUTABLE CODE FILES</p> <p>THE OM TEXT ENTRIES MAY BE READY TO RUN OR NEED SOME PREPARATION, BUT IN ANY CASE THE LOCATION OF THE CODE MUST BE KNOWN FOR CONTROL CARD FILL IN.</p>
-------	--

N.C.D	<p>CREATE CONTROL CARDS FOR THIS EXECUTION</p> <p>TAKE ALL THE EXECUTION TIME DEPENDENT INFORMATION AND PLACE/SUBSTITUTE IT IN THE CONTROL CARD SKELETON.</p>
-------	---

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→N.C.A	N.C.B	1	1
N.C.B	N.C.C	2	2
N.C.C	N.C.D	3	3
N.C.D	→N.D.A	4	4



LEVEL 3 TRANSITION DIAGRAM

STATE N.C : PREPARE JOB FOR EXECUTION

N.C : PREPARE JOB FOR EXECUTION
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	OM NETWORK AND THE LIST OF LEN USED DURING EXECUTION
2	CORRESPONDANCE LIST FOR EACH OM GIVING THE RELATIONSHIP BETWEEN LEN AND LOGICAL FILE NAME USED IN THE OS
3	OM TEXT ENTRY AND THE CORRESPONDING TEXT LOCATION SPECIFICATIONS.
4	COMPLETE STATUS ON CONTROL CARD RECORD

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	NAMES OF ALL LE USED FOR THIS EXECUTION
2	CONSISTENT FILE DEFINITIONS FOR ENTIRE DATA FLOW
3	EXECUTABLE CODE FILES.
4	ALL JOB COMPONENTS READY FOR EXECUTION

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.C.A	N.B.A N.B.B

LEVEL 3
COMPONENTS OF STATE N.D
INITIATE EXECUTION

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
-------	--------------------

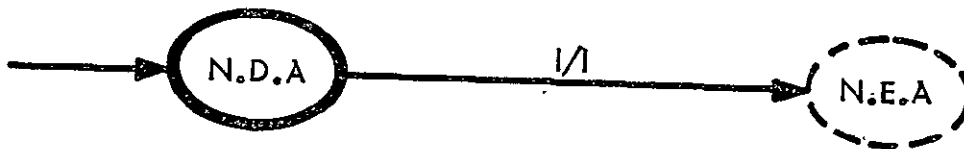
N.D.A	INITIATE EXECUTION
-------	--------------------

ISSUE A COMMAND TO THE OS TO EXECUTE THE CONTROL
CARD RECORD AND WAIT UNTIL THE EXECUTION COMMAND HAS
BEEN ACCEPTED.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P.N.D.A	P.N.E.A	1	1

TRANSITION DIAGRAM



N.D : INITIATE EXECUTION
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SIGNAL FROM THE OS THAT THE JOB EXECUTION COMMAND HAS BEEN ACCEPTED AND THE JOB IS EXECUTING.

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.D.A	N.C.D

LEVEL 3
COMPONENTS OF STATE N.E
SUBTASK STEP EXECUTING

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

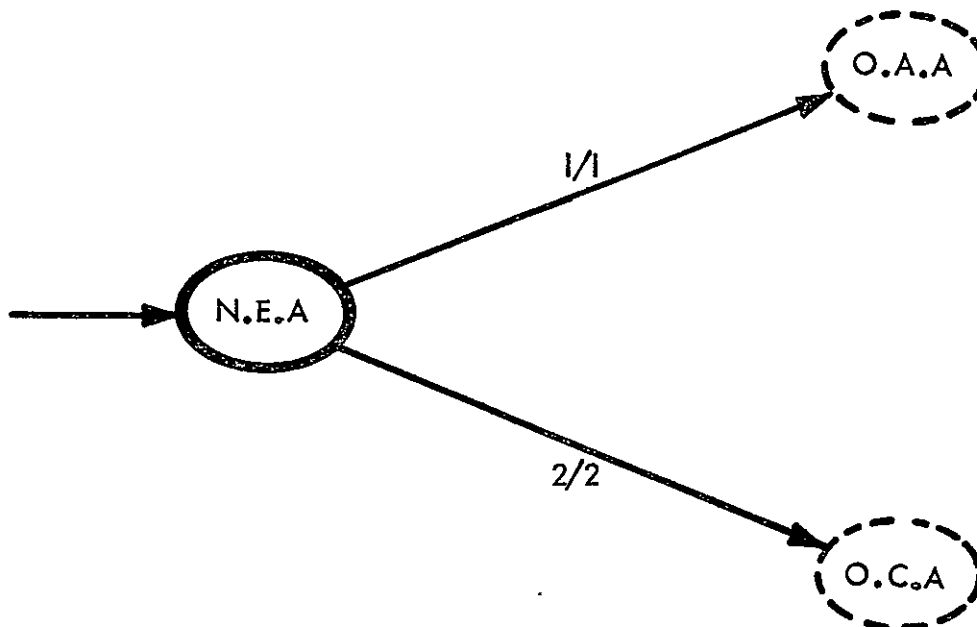
N.E.A SUBTASK STEP EXECUTING

SOME ARBITRARY PORTION OF THE SUBTASK STEP IS NOW
EXECUTING AND INDICATES NO REQUIREMENT FOR USER OR IPA
EXECUTIVE INTERACTION.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
PN.E.A	PO.A	1	1
	PO.C	2	2

TRANSITION DIAGRAM



N.E : SUBTASK STEP EXECUTING
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	INPUT REQUEST COMMAND FROM THE STS
2	OUTPUT COMMAND FROM THE STS

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	TERMINAL INPUT REQUEST
2	TERMINAL OUTPUT REQUEST

***** CROSS REFERENCED TRANSITIONS *****

STATE	IS ACCESSIBLE FROM
N.E.A	N.D.A

LEVEL 3
COMPONENTS OF STATE Q.C
PURGE A CL ENTRY

***** STATE DESCRIPTIONS *****

ORIGINAL PAGE IS
OF POOR QUALITY

STATE LONG NAME AND TEXT

Q.C.A RETRIEVE DICTIONARY ENTRY

Q.C.B RETRIEVE DIRECTORY ENTRY

THE SUBTASK ID LIST MUST BE EMPTY BEFORE A PURGE
CAN BE EFFECTED.

Q.C.C TRACE DEPENDENCIES

THE DICTIONARY AND DIRECTORY ENTRIES ARE CHECKED
FOR DEPENDENCIES. IF THE USED-BY LIST IN THE DICTIONARY
IS NOT EMPTY THE DEPENDENCY CHAINS ARE TRACED. THE SUB-
TASK ID LIST IN THE DIRECTORY ENTRY MUST ALSO BE EMPTY
BEFORE A PURGE IS PERMISSIBLE.

Q.C.D PUBLISH DEPENDENCY LIST

A COMPLETE LIST OF ALL DEPENDENT LIBRARY ENTRIES
AND THE OWNER OF EACH IS PUBLISHED

Q.C.E SET STATUS

THE STATUS IS SET = *PURGE REQUESTED* IF DEPEND-
ENCIES WERE FOUND. STATUS IS SET = *PURGED* IF DATA IS
RELEASED FROM DATA BASE.

Q.C.F PURGE DATA

RELEASE DATA FROM TEXT OF LE.

Q.C : PURGE A CL ENTRY
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

STATE . . . LONG NAME AND TEXT

Q.C.G CLEAR REFERENCES

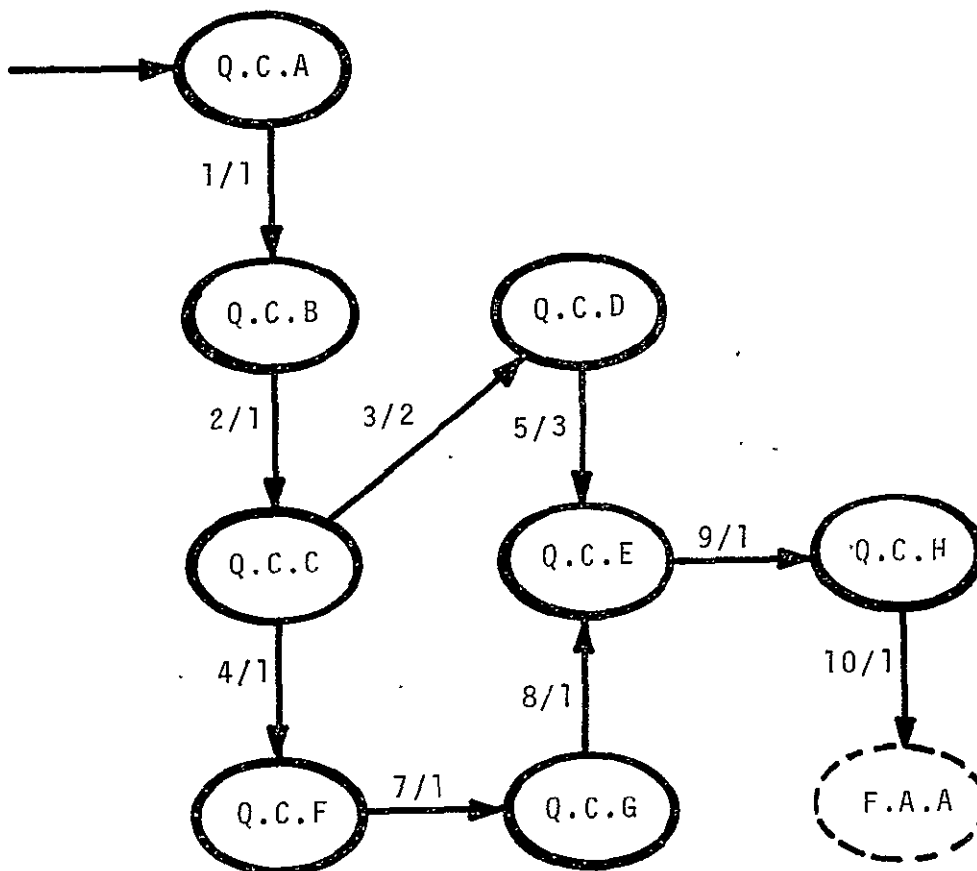
REFERENCES IN EXISTING LIBRARY ENTRIES WHICH SHOW
PURGED LE AS A USER ARE CLEARED.

Q.C.H CLEAR DIRECTORY ENTRY

THE DIRECTORY ENTRY IS CLEARED OF ALL INFORMATION
EXCEPT NAME,TYPE, FINAL STATUS RECORD.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→Q.C.A	Q.C.B	1	1
Q.C.B	Q.C.C	2	1
Q.C.C	Q.C.D	3	2
	Q.C.F	4	1
Q.C.D	Q.C.E	5	3
Q.C.E	Q.C.H	9	1
Q.C.F	Q.C.G	7	1
Q.C.G	Q.C.E	6	1
Q.C.H	→F.A.A	10	1



LEVEL 3 TRANSITION DIAGRAM
STATE Q.C: PURGE A CL ENTRY

Q.C : PURGE A CL ENTRY
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	DICTIONARY ENTRY RETRIEVED
2	DIRECTORY ENTRY RETRIEVED
3	DEPENDENCIES FOUND
4	NO DEPENDENCIES FOUND
5	LIST OF DEPENDENCIES PUBLISHED
7	DATA FROM TEXT ENTRY RELEASED FROM DATA BASE
8	ALL REFERENCES TO PURGED LE CLEARED FROM DATA BASE
9	STATUS SET = *PURGED*
10	DIRECTORY ENTRY CLEARED OF ALL BUT FINAL STATUS

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	ERROR MESSAGE
3	ERROR CONDITION TO BE CLEARED

LEVEL 3
COMPONENTS OF STATE W.C
CONSTRUCT DICTIONARY ENTRY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

W.C.A RETRIEVE SDD FOR DICTIONARY

THE SDD FOR LE TYPE DICTIONARY IS RETRIEVED FOR
USE IN SUBSEQUENT PROCESSING.

W.C.B ENTER DICTIONARY INFORMATION

THE USER ENGAGES IN A DIALOGUE WITH THE SYSTEM
DURING WHICH THE INFORMATION FOR THE DICTIONARY ENTRY
IS PROVIDED.

W.C.C REVIEW ENTRY

THE NEW DICTIONARY ENTRY IS DISPLAYED TO AND RE-
VIEWED BY THE USER.

W.C.D ADD ENTRY TO DICTIONARY

THE NEW ENTRY HAS BEEN APPROVED BY THE USER AND
IS ADDED TO THE DICTIONARY IN THE DATA BASE

W.C.E ERROR RECOVERY

RESOLUTION OF A CONFLICT OVER AN ENTRY IN A CL
DICTIONARY IS MADE. AN ENTRY HAVING SAME NAME,TYPE WAS
MADE BY ANOTHER USER IN THE TIME INTERVAL AFTER THE
FIRST CHECK MADE AT THE TIME DIALOGUE BEGAN AND THE
TIME THE USER WAS READY TO ADD THE ENTRY.

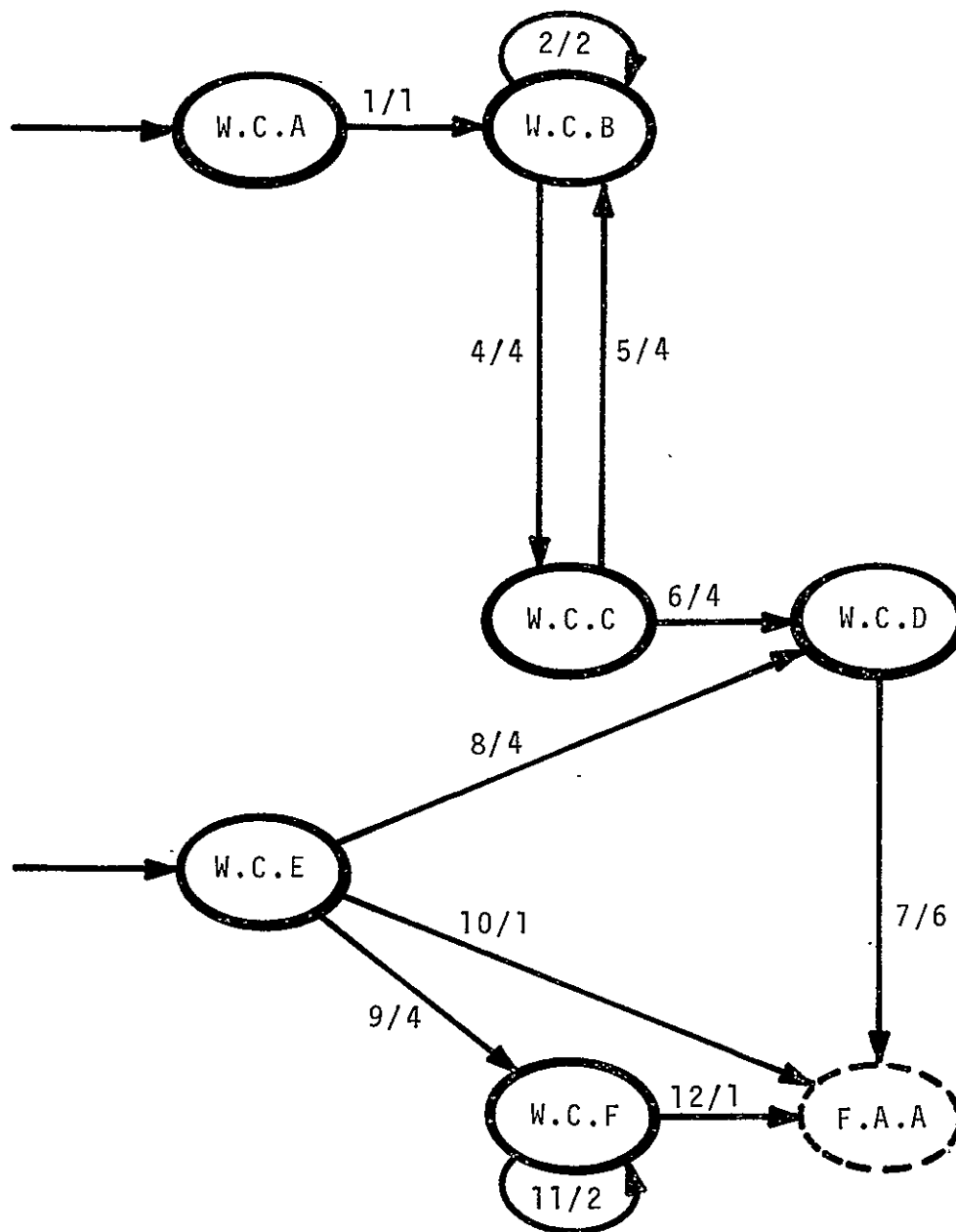
W.C.F SAVE ENTRY TEXT IN SL

THE USER HAS BEEN UNABLE TO SATISFACTORILY RESOLVE
A NAME CONFLICT AND SAVES THE ENTRY LOCALLY.

W.C : CONSTRUCT DICTIONARY ENTRY
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→W.C.A	W.C.B	1	1
W.C.B	W.C.B	2	2
	W.C.C	4	4
W.C.C	W.C.B	5	4
	W.C.D	6	4
W.C.D	→F.A.A	7	6
→W.C.E	→F.A.A	10	1
	W.C.U	8	4
	W.C.F	9	4
W.C.F	→F.A.A	12	1
	W.C.F	11	2



LEVEL 3 TRANSITION DIAGRAM

STATE W.C: CONSTRUCT DICTIONARY ENTRY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDJ FOR DICTIONARY RETRIEVED
2	USER/SYSTEM DIALOG INCOMPLETE
4	DICTIONARY ENTRY COMPLETE BUT NOT APPROVED
5	USER WANTS TO MODIFY ENTRY
6	DICTIONARY ENTRY COMPLETE AND APPROVED
7	ENTRY ADDED TO DICTIONARY
3	NAME CONFLICT RESOLVED
9	CONFLICT NOT RESOLVED-USER WANTS TO SAVE ENTRY TEXT LOCALLY
10	USER WANTS NO FURTHER ACTION ON THIS ENTRY
11	USER/SYSTEM DIALOGUE SPECIFYING LOCAL SAVE OF TEXT
12	ENTRY SAVED IN SL AS SPECIFIED BY USER.

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER REQUESTING MORE INFORMATION
4	DICTIONARY ENTRY TEXT IN USER WORKING AREA
6	UPDATED DICTIONARY AVAILABLE IN DATA BASE

LEVEL 4
COMPONENTS OF STATE I.C.B
ENTER CODING MODULE

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.C.B.A CHECK LIBRARY ENTRY

THE APPROPRIATE LIBRARY DIRECTORY IS SEARCHED FOR A CODING MODULE HAVING THE SAME NAME. IF ONE IS FOUND AN ERROR CONDITION EXISTS AND IS REPORTED.

I.C.B.B CHECK DICTIONARY ENTRY

THE APPROPRIATE DICTIONARY IS SEARCHED FOR A PREVIOUSLY MADE DEFINITION. IF WORKING IN THE CL AND ONE IS FOUND IT IS OFFERED FOR REVIEW. IF IN THE CL AND NO DICTIONARY ENTRY EXISTS, ONE MUST BE MADE. IF IN THE STL THE DE IS OPTIONAL.

I.C.B.C REVIEW DICTIONARY ENTRY

USER MAY DESIRE TO EXAMINE EXISTING DEFINITION FOR CORRECTNESS. IF A CHANGE IS DESIRED HE ENTERS THE MODIFY DEFINITION MODE VIA A PAUSE.

I.C.B.D DEFINE CM

A DEFINITION FOR THE NEW CM IS ENTERED (A DICTIONARY ENTRY MADE)

I.C.B.E RETRIEVE SDD FOR CM

I.C.B.F REPORT ERROR

I.C.B.G CONSTRUCT CM

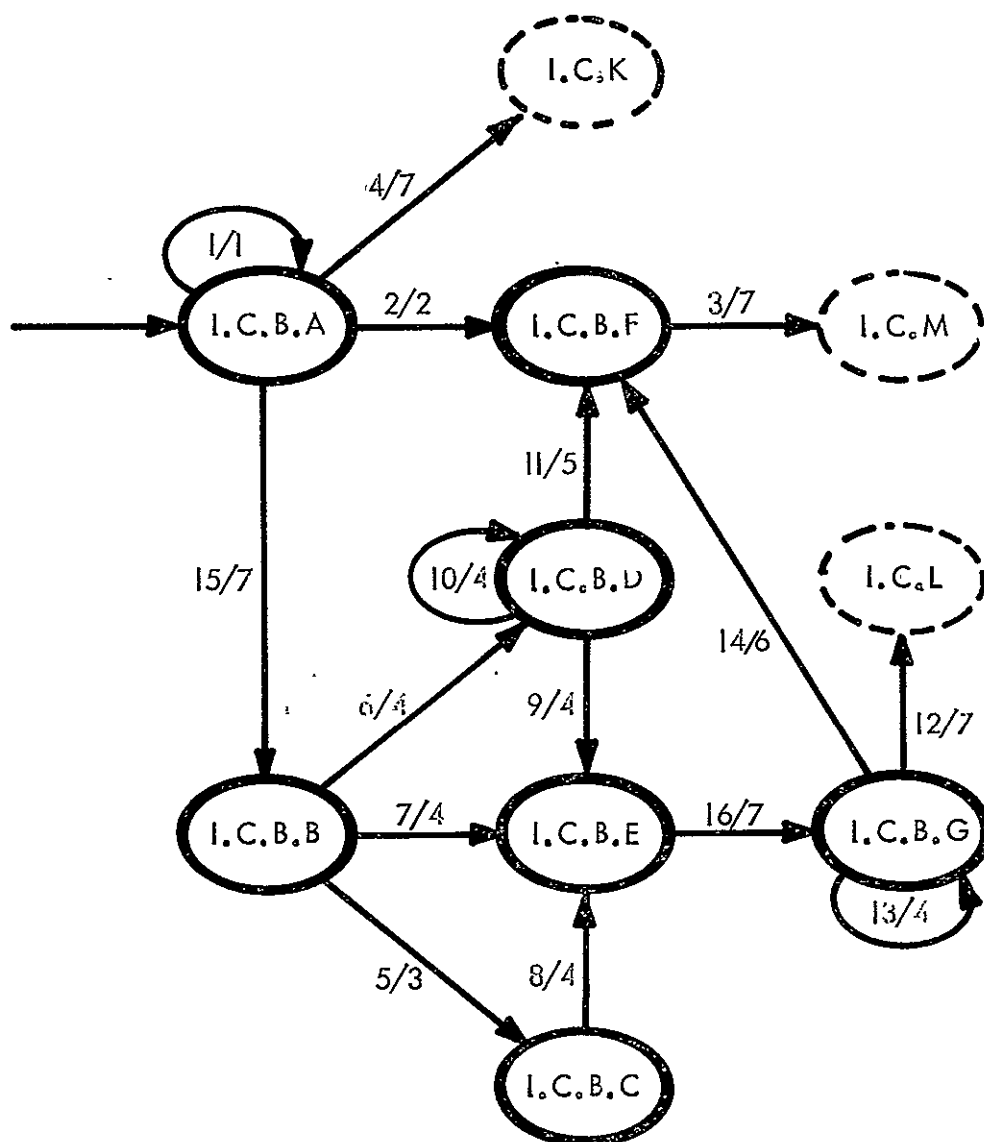
THE SOURCE CODE AND AUXILIARY DATA WHICH COMPRISE THE LIBRARY ENTRY FOR THE CM ARE ENTERED

I.C.B : ENTER CODING MODULE
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→I.C.B.A	I.C.B.A	1	1
	I.C.B.B	15	7
	I.C.B.F	2	2
	→I.C.K	4	7
I.C.B.B	I.C.B.C	5	3
	I.C.B.D	6	4
	I.C.B.E	7	4
I.C.B.C	I.C.B.E	8	4
I.C.B.D	I.C.B.D	10	4
	I.C.B.E	9	4
	I.C.B.F	11	5
I.C.B.E	I.C.B.G	16	7
I.C.B.F	→I.C.M	3	7
I.C.B.G	I.C.B.F	14	6
	I.C.B.G	13	4
	→I.C.L	12	7

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 4 TRANSITION DIAGRAM

STATE I.C.B: ENTER CODING MODULE

I.C.8 : ENTER CODING MODULE
(CONTINUED.)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	A CM WITH SAME NAME AS NEW CM FOUND IN LIBRARY
2	AMBIGUOUS CM NAMES NOT RESOLVED
3	ERROR CONDITION POSTED
4	NO PREVIOUS LE FOR CM EXISTS
5	PREVIOUS DICTIONARY ENTRY EXISTS
6	NO DICTIONARY ENTRY EXISTS AND ONE IS REQUIRED OR DESIRED
7	NO DICTIONARY ENTRY EXISTS AND ONE NOT REQUIRED OR WANTED
8	DICTIONARY ENTRY APPROVED
9	DICTIONARY COMPLETE AND IN DATA BASE
10	MORE INFORMATION REQUIRED TO COMPLETE DICTIONARY ENTRY
11	IMPOSSIBLE TO COMPLETE DICTIONARY ENTRY
12	ALL DATA FOR CM LE ACQUIRED
13	MORE DATA REQUIRED FOR LE
14	IMPOSSIBLE TO COMPLETE LE
15	DIRECTORY ENTRY FOR NEW LE ESTABLISHED IN USER WORKING AREA
16	SDD RETRIEVED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE TO USER GIVING OPORTUNITY TO REMOVE AMBIGUITY
2	ERROR MESSAGE (AMBIGUOUS CM NAME)
3	TEXT OF DICTIONARY ENTRY AVAILABLE FOR DISPLAY
4	MESSAGE INFORMING USER TO PROCEED
5	ERROR MESSAGE (DE NOT COMPLETED)
6	ERROR MESSAGE (LE NOT COMPLETED)
7	-

ORIGINAL PAGE IS
OF POOR QUALITY

LEVEL 4
COMPONENTS OF STATE I.C.C
ENTER DATA SET

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.C.C.A RETRIEVE STORED DATA DEFINITION

THE USER WANTS TO CREATE AN INSTANCE OR OCCURRENCE OF A DATA SET BY ENTERING VALUES FOR VARIABLES CONTAINED IN THE DATA SET. SUBSEQUENT PROCESSING REQUIRES AN SDD. IF THERE IS NO SDD IN HIS STL OR IN THE CL WHICH THE USER HAS PERMISSION TO USE AN ERROR CONDITION IS ESTABLISHED PREVENTING CONTINUATION AT THIS POINT.

I.C.C.B CONSTRUCT TEXT FOR NEW LE

THE USERS DATA IS ACCEPTED, TRANSFORMED, FORMATTED AND PREPARED FOR STORAGE FOLLOWING SPECIFICATIONS CONTAINED IN THE SDD. THE DATA MAY COME FROM THE TERMINAL OR FROM OTHER ORIGINS EXTERNAL TO IPAD SUCH AS MAGNETIC TAPE, PUNCHED CARDS, DISK FILES, AS INDICATED BY THE USER. DATA FROM SEVERAL SOURCES MAY BE COMBINED.

I.C.C.C REPORT ERROR

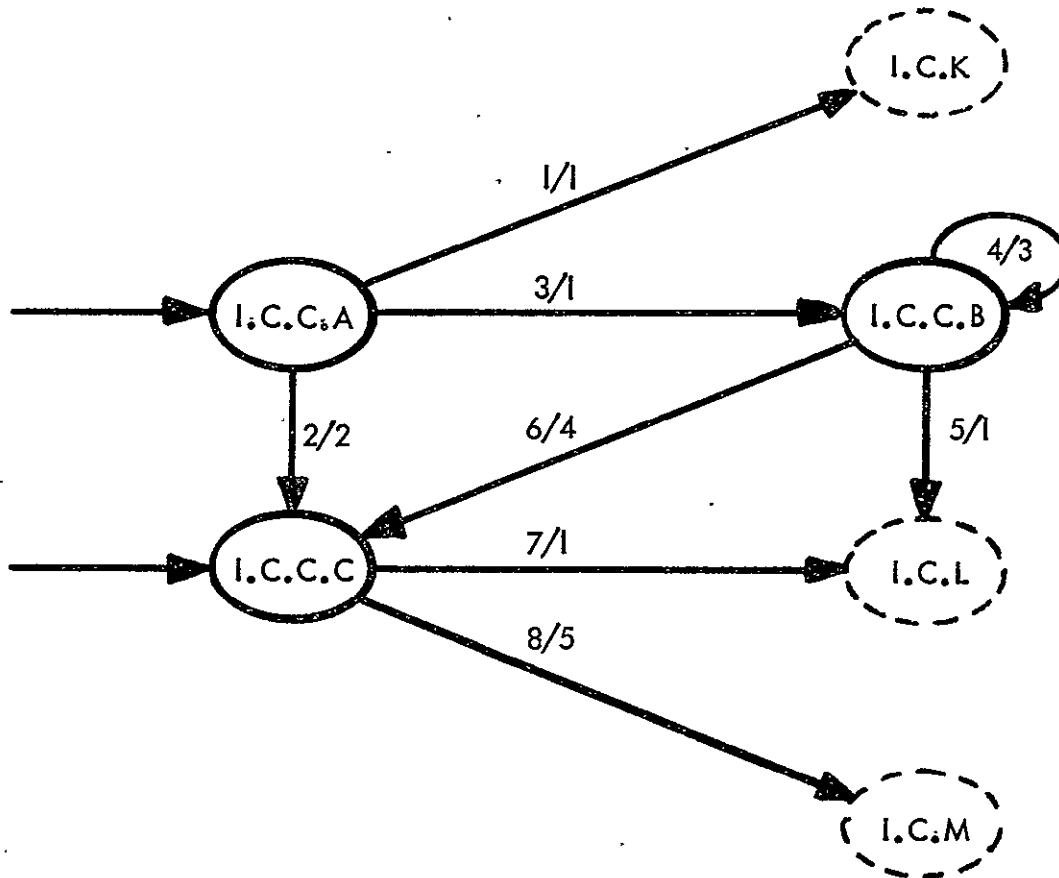
SOME ERROR CONDITIONS MAY OCCUR AFTER MUCH VALUABLE PROCESSING HAS OCCURRED IN WHICH CASE THE USER MAY ELECT TO SAVE THE DATA ENTERED FOR LATER CORRECTION VIA MODIFY.

I.C.C : ENTER DATA SET
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→ I.C.C.A	I.C.C.B	3	1
	I.C.C.C	2	2
	→ I.C.K	1	1
I.C.C.B	I.C.C.B	4	3
	I.C.C.C	6	4
→ I.C.C.C	→ I.C.L	5	1
	→ I.C.L	7	1
	→ I.C.M	8	5

TRANSITION DIAGRAM



I.C.C : ENTER DATA SET
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	STORED DATA DEFINITION CORRESPONDING TO NEW LE EXISTS
2	SDD DOES NOT EXIST
3	DIRECTORY ENTRY FOR NEW LE ESTABLISHED IN USER WORKING AREA
4	DATA INPUT AND LE CONSTRUCTION INCOMPLETE
5	LE CONSTRUCTION COMPLETE IN USER WORKING AREA
6	IMPOSSIBLE TO PROCESS SOME SPECIFIC INPUT DATA
7	USER DESIRES TO SAVE DATA SET AS IS
3	ERROR CONDITION TO BE CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	ERROR MESSAGE (SDD DOES NOT EXIST)
3	MESSAGE REQUESTING USER TO ENTER MORE DATA
4	ERROR MESSAGE (DATA PROCESSING ERROR)_____
5	APPROPRIATE ERROR CODE

LEVEL 4
COMPONENTS OF STATE I.C.D
ENTER STORED DATA DEFINITION

ORIGINAL PAGE IS
OF POOR QUALITY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.C.D.A CHECK LIBRARY ENTRY

THE APPROPRIATE LIBRARY DIRECTORY IS SEARCHED FOR AN SDD HAVING THE SAME NAME. IF ONE IS FOUND AN ERROR CONDITION EXISTS AND IS REPORTED.

I.C.D.B CHECK DICTIONARY ENTRY

THE APPROPRIATE DICTIONARY IS SEARCHED FOR AN EXISTING DEFINITION. IF WORKING IN THE CL AND ONE IS FOUND IT IS OFFERED FOR REVIEW. IF WORKING IN THE CL AND NO DICTIONARY ENTRY EXISTS, ONE MUST BE MADE. IF IN THE STL THE DE IS OPTIONAL.

I.C.D.C REVIEW DICTIONARY ENTRY

USER MAY DESIRE TO EXAMINE EXISTING DEFINITION FOR CORRECTNESS. IF A CHANGE IS DESIRED HE ENTERS THE MODIFY DEFINITION MODE VIA A PAUSE

I.C.D.D DEFINE SDD

A DICTIONARY ENTRY FOR THE DATA SET IS MADE.

I.C.D.E RETRIEVE SDD FOR SDD

THE SDD WHICH SPECIFIES THE STRUCTURE IN WHICH ALL STORED DATA DEFINITIONS ARE STORED IN IPAD IS RETRIEVED.

I.C.D.F REPORT ERROR

I.C.D : ENTER STORED DATA DEFINITION
(CONTINUED)

***** STATE DESCRIPTIONS (CONTINUED) *****

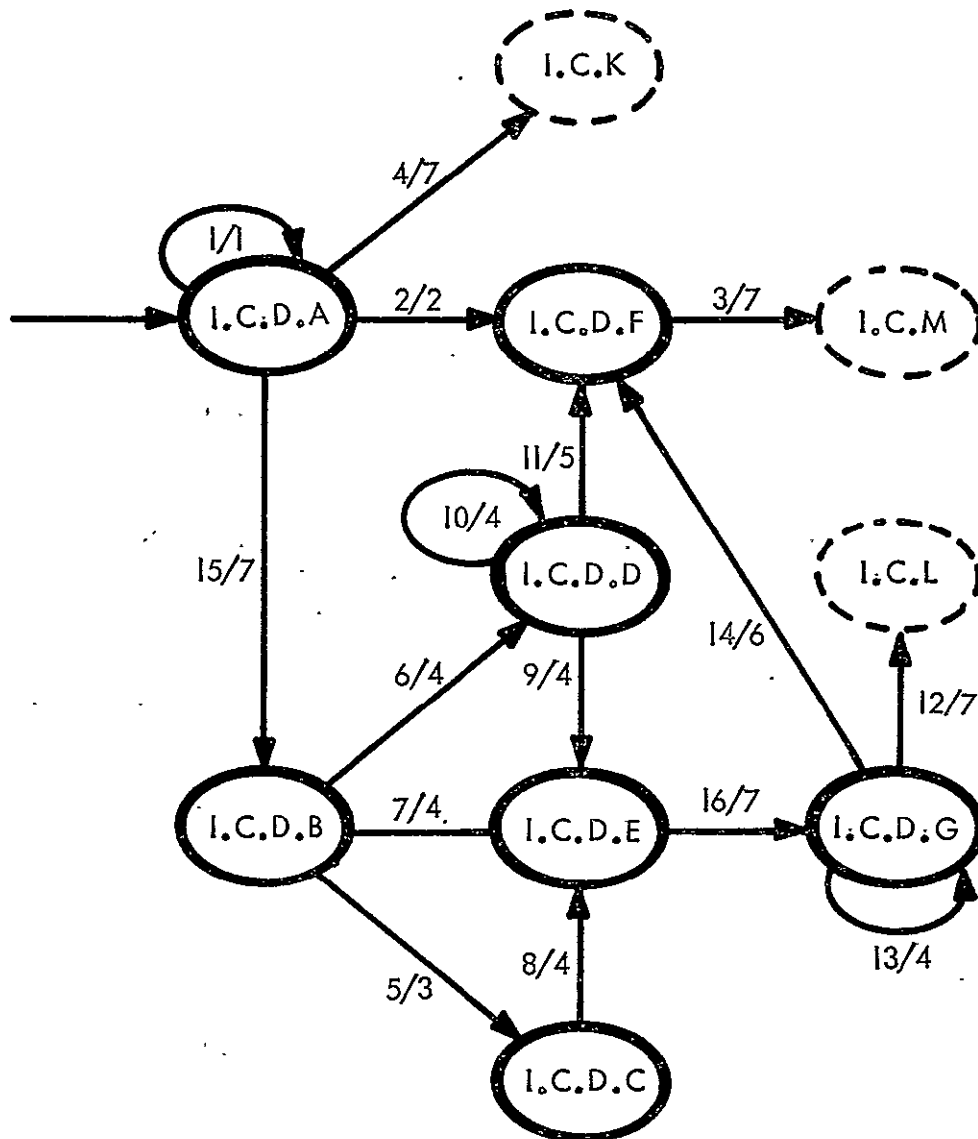
STATE LONG NAME AND TEXT

I.C.D.G CONSTRUCT SDD

THE INFORMATION WHICH SPECIFIES THE CONTENTS
AND ORGANIZATION OF THE DATA SETS FOR WHICH THIS
SDD IS TO BE USED IS ENTERED AND A LE OF TYPE SDD
CONSTRUCTED.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P I.C.D.A	I.C.D.A	1	1
	I.C.D.B	15	7
	I.C.D.F	2	2
	P I.C.K	4	7
I.C.D.B	I.C.D.C	5	3
	I.C.D.D	6	4
	I.C.D.E	7	4
I.C.D.C	I.C.D.E	8	4
I.C.D.D	I.C.D.D	10	4
	I.C.D.E	9	4
	I.C.D.F	11	5
I.C.D.E	I.C.D.G	16	7
I.C.D.F	P I.C.M	3	7
I.C.D.G	I.C.D.F	14	6
	I.C.D.G	13	4
	P I.C.L	12	7



LEVEL 4 TRANSITION DIAGRAM

STATE I.C.D : ENTER SDD

I.C.D : ENTER STORED DATA DEFINITION
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	AN SDD WITH SAME NAME AS NEW SDD FOUND IN LIBRARY
2	AMBIGUOUS NAMES NOT RESOLVED
3	ERROR CONDITION POSTED
4	NO PREVIOUS LE FOR SDD EXISTS
5	PREVIOUS DICTIONARY ENTRY EXISTS
6	NO DICTIONARY ENTRY EXISTS AND ONE IS REQUIRED OR DESIRED
7	NO DICTIONARY ENTRY EXISTS AND ONE IS NEITHER RE- QUIRED NOR WANTED.
8	DICTIONARY ENTRY APPROVED
9	DICTIONARY COMPLETE AND IN DATA BASE
10	MORE INFORMATION REQUIRED TO COMPLETE DICTIONARY ENTRY
11	IMPOSSIBLE TO COMPLETE DICTIONARY ENTRY
12	ALL DATA FOR SDD ACQUIRED
13	MORE DATA FOR SDD REQUIRED
14	IMPOSSIBLE TO COMPLETE LE FOR NEW SDD
15	DIRECTORY ENTRY FOR NEW LE ESTABLISHED IN USER WORKING AREA
16	SDD FOR SDD RETRIEVED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE TO USER GIVING OPPORTUNITY TO REMOVE AMBIGUITY
2	ERROR MESSAGE (AMBIGUOUS SDD NAME)
3	TEST OF DICTIONARY ENTRY AVAILABLE FOR DISPLAY
4	MESSAGE INFORMING USER TO PROCEED
5	ERROR MESSAGE (OE NOT COMPLETED)
6	ERROR MESSAGE (LE NOT COMPLETED)
7	-

LEVEL 4
COMPONENTS OF STATE I.C.E
ENTER DICTIONARY

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.C.E.A CHECK LIBRARY ENTRY

THE USER CAN ONLY DEFINE NEW DICTIONARIES FOR HIS STL. TWO DICTIONARIES FOR STL VARIABLES AND LIBRARY ENTRIES ARE GIVEN DEFAULT NAMES EQUAL TO THE CL DICTIONARY NAMES WITH SUBTASK ID APPENDED. THE USER IS FREE TO DEFINE ADDITIONAL DICTIONARIES TO SUIT HIS OWN NEEDS BUT THE DEFAULT DICTIONARIES ARE USED FOR SUBTASKS IN A MANNER ANALOGOUS TO THE CL USAGE.

I.C.E.B CHECK DICTIONARY ENTRY

THE LOCAL OR SL DICTIONARY IS CHECKED FOR PREVIOUS DEFINITION OF NON-DEFAULT NAMED DICTIONARY.

I.C.E.C REVIEW DICTIONARY ENTRY

I.C.E.D DEFINE NEW DICTIONARY

I.C.E.E RETRIEVE SDD FOR DICTIONARY

I.C.E.F REPORT ERROR

I.C.E.G CONTRUCT DICTIONARY

0 100 0 0 0 0
0 100 0 0 0 0

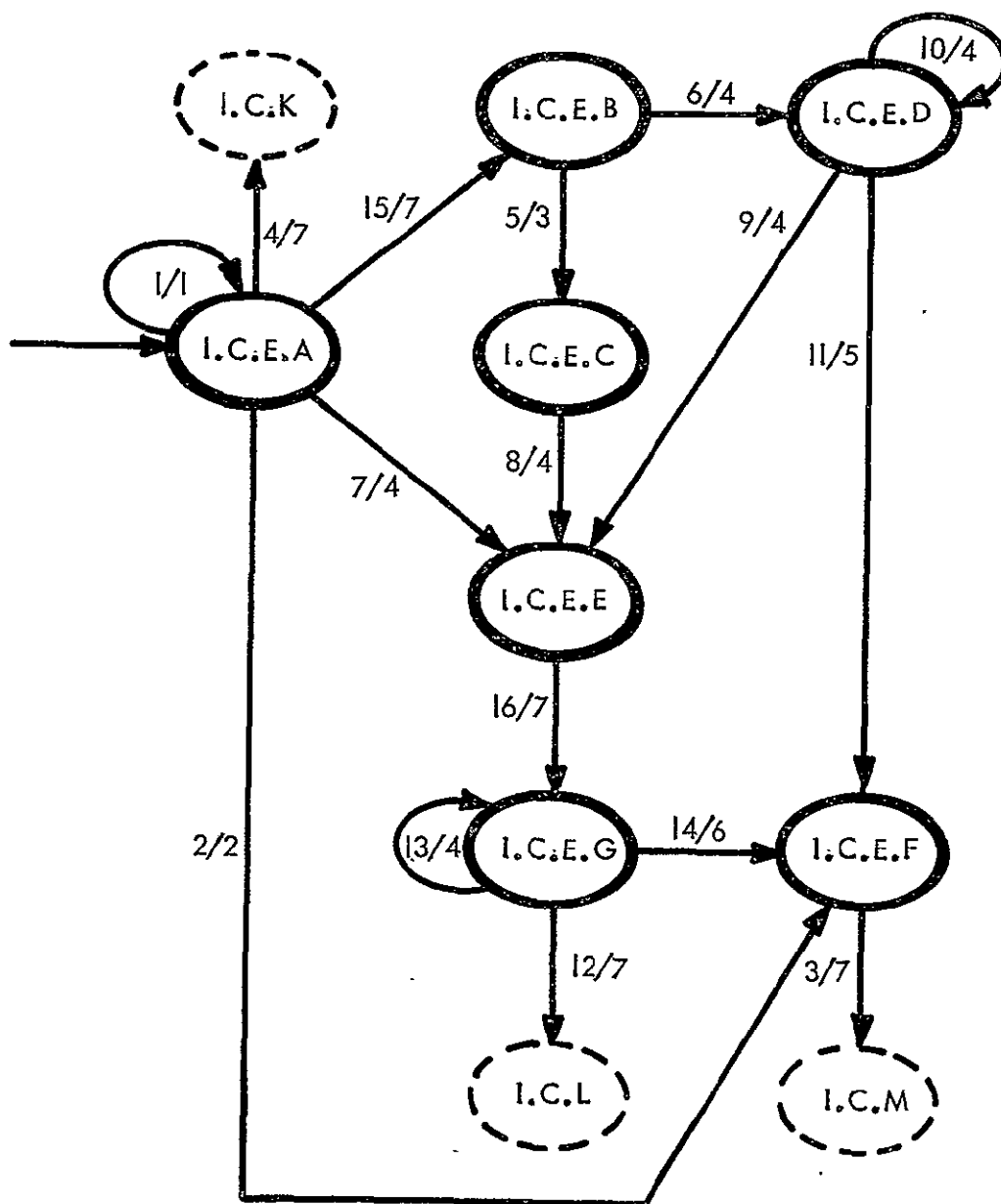
I.C.E : ENTER DICTIONARY
(CONTINUED)

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→I.C.E.A	I.C.E.A	1	1
	I.C.E.B	15	7
	I.C.E.E	7	4
	I.C.E.F	2	2
	→I.C.K	4	7
I.C.E.B	I.C.E.C	5	3
	I.C.E.D	6	4
I.C.E.C	I.C.E.E	3	4
I.C.E.D	I.C.E.D	13	4
	I.C.E.E	9	4
	I.C.E.F	11	5
I.C.E.E	I.C.E.G	16	7
I.C.E.F	→I.C.M	3	7
I.C.E.G	I.C.E.F	14	6
	I.C.E.G	13	4
	→I.C.L	12	7

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 4 TRANSITION DIAGRAM

STATE I.C.E : ENTER DICTIONARY

I.C.E : ENTER DICTIONARY
(CONTINUED)

ORIGINAL PAGE IS
OF POOR QUALITY

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	A DICTIONARY WITH SAME NAME AS NEW DICTIONARY FOUND IN STL
2	AMBIGUOUS DICTIONARY NAMES NOT RESOLVED
3	ERROR CONDITION POSTED
4	NO PREVIOUS LE FOR DICTIONARY
5	PREVIOUS DICTIONARY ENTRY EXISTS
6	NO DICTIONARY ENTRY EXISTS
7	USER WANTS DEFAULT NAMEJ LE DICTIONARY
8	DICTIONARY ENTRY APPROVED
9	DICTIONARY ENTRY COMPLETE AND IN DATA BASE
10	MORE INFORMATION REQUIRED TO COMPLETE DICTIONARY ENTRY
11	IMPOSSIBLE TO COMPLETE DICTIONARY ENTRY
12	ALL DATA FOR DICTIONARY LE ACQUIRED
13	MORE DATA REQUIRED FOR LE
14	IMPOSSIBLE TO COMPLETE LE
15	DIRECTORY ENTRY FOR NEW LE ESTABLISHED IN USER WORKING AREA
16	SUD RETRIEVED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	MESSAGE TO USER GIVING OPPORTUNITY TO REMOVE AMBIGUITY
2	ERROR MESSAGE (AMBIGUOUS DICTIONARY NAME)
3	TEXT OF DICTIONARY ENTRY(DEFINING NEW DICTIONARY) IS AVAILABLE FOR DISPLAY
4	MESSAGE INFORMING USER TO PROCEED
5	ERROR MESSAGE (DIRECTORY ENTRY NOT COMPLETED)
6	ERROR MESSAGE (LIBRARY ENTRY NOT COMPLETED)
7	-

LEVEL 4
COMPONENTS OF STATE I.C.J
ENTER DATA CONTROL DATA

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

I.C.J.A RETRIEVE SDD

THE SDD FOR THE SYSTEM DATA STRUCTURE WHICH HOLDS
THE SECURITY, ACCESS DATA IS RETRIEVED FOR USE IN THE
PROCESSING OF THE INPUT DATA

I.C.J.B CONSTRUCT TEXT

THE CONTROL DATA IS ACCEPTED, PROCESSED AND PRE-
PARED FOR STORAGE.

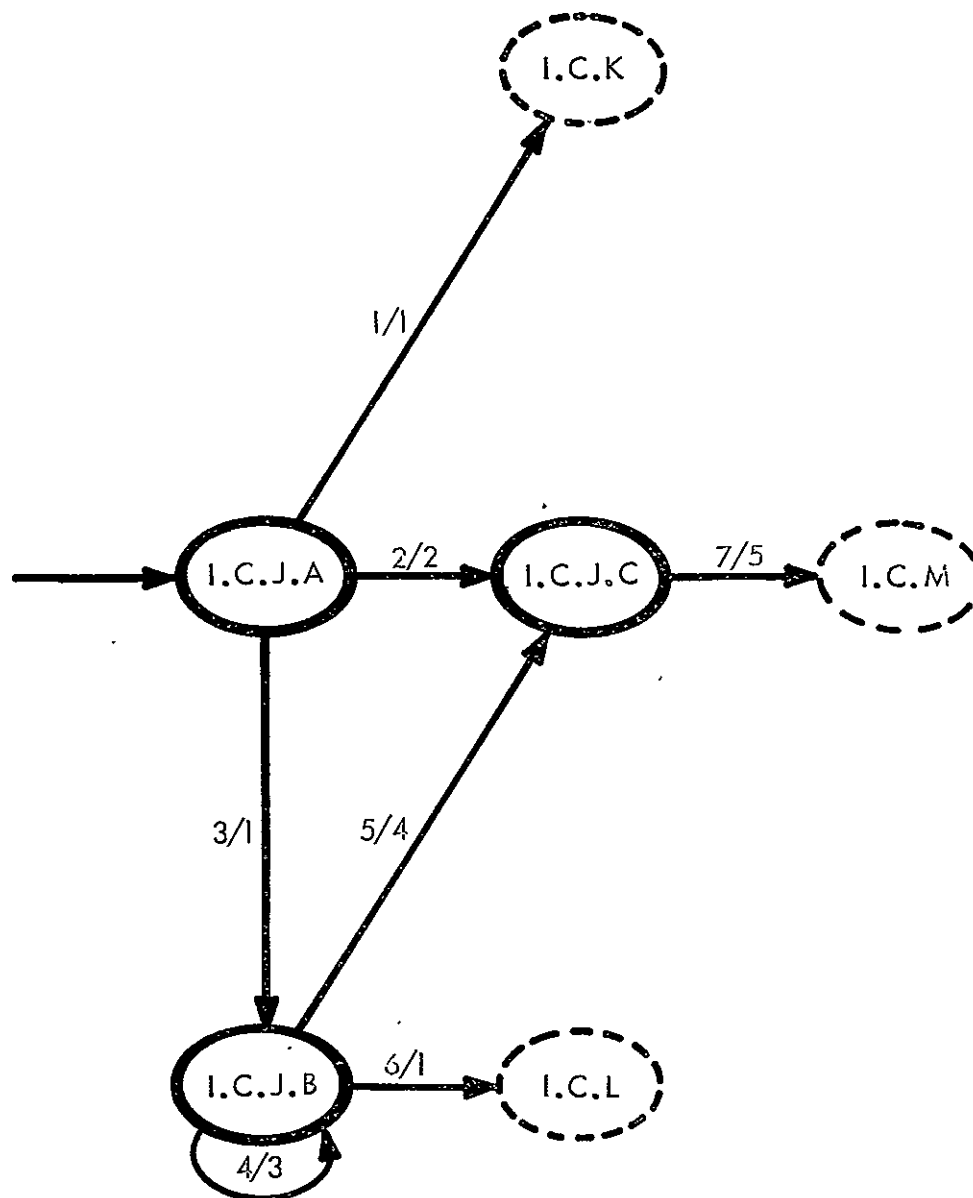
I.C.J.C REPORT ERROR

ALL ERROR CONDITIONS ENCOUNTERED IN THIS STATE ARE
FATAL. PROCESSING IS ABORTED.

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
P I.C.J.A	I.C.J.B	3	1
	I.C.J.C	2	2
I.C.J.B	P I.C.K	1	1
	I.C.J.B	4	3
	I.C.J.C	5	4
I.C.J.C	P I.C.L	6	1
	P I.C.M	7	5

ORIGINAL PAGE IS
OF POOR QUALITY



LEVEL 4 TRANSITION DIAGRAM

STATE I.C.J : ENTER DATA CONTROL DATA

I.C.J : ENTER DATA CONTROL DATA
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDD FOR SYSTEM LE TYPE DCU EXISTS
2	SDD DOES NOT EXIST
3	DIRECTORY ENTRY FOR CONTROL DATA LE ESTABLISHED IN USER WORKING AREA
4	DATA INPUT AND LE CONSTRUCTION INCOMPLETE
5	IMPOSSIBLE TO PROCESS SOME SPECIFIC INPUT DATA
6	LE CONSTRUCTION COMPLETE IN USER WORKING AREA
7	ERROR CONDITION TO BE CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	ERROR MESSAGE
3	MESSAGE REQUESTING USER TO ENTER MORE DATA
4	ERROR MESSAGE
5	APPROPRIATE ERROR CODE

LEVEL 4
COMPONENTS OF STATE K.B.A
MODIFY CM

***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
-------	--------------------

K.B.A.A	RETRIEVE STORED DATA DEFINITION
---------	---------------------------------

THE SDD FOR A CM IS RETRIEVED FOR USE IN FOLLOWING
PROCESSING.

K.B.A.B	PERFORM C4 MODIFICATIONS
---------	--------------------------

THE USER ENGAGES IN DIALOGUE WITH THE SYSTEM TO
CARRY OUT HIS CHANGES.

K.B.A.C	REPORT ERROR
---------	--------------

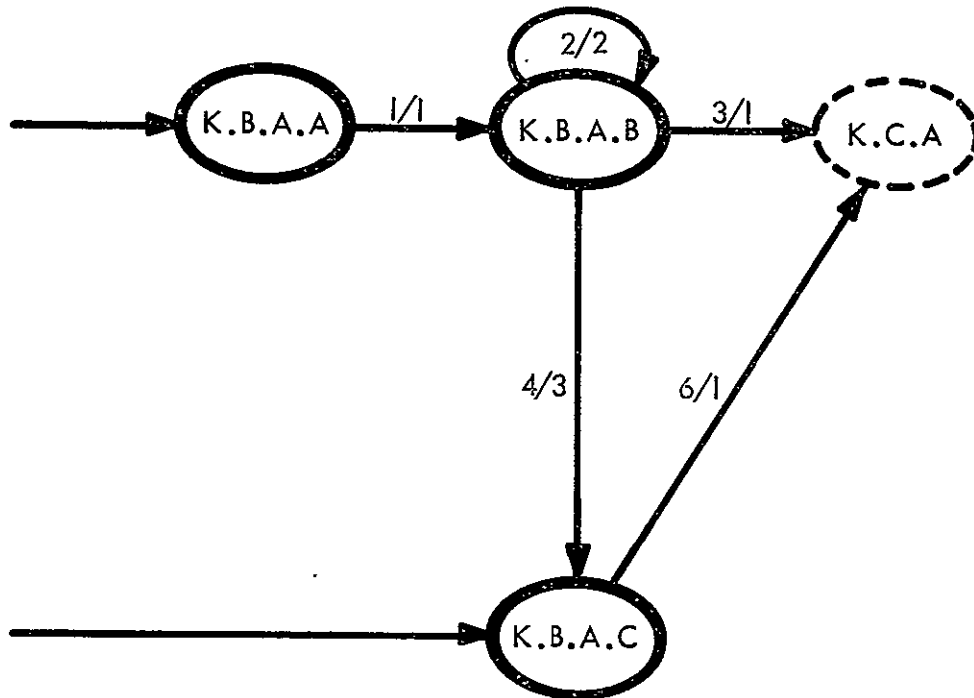
***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.B.A.A	K.B.A.B	1	1
K.B.A.B	K.B.A.B	2	2
	K.B.A.C	4	3
→K.B.A.C	→K.C.A	3	1
	→K.C.A	6	1

K.B.A : MODIFY CM
(CONTINUED)

FINAL PAGE IN
POOR QUALITY

TRANSITION DIAGRAM



***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDJ FOR CM RETRIEVED
2	MODIFICATIONS INCOMPLETE
3	MODIFICATIONS COMPLETE
4	ERROR CONDITION ENCOUNTERED DURING MODIFICATION
5	ERROR CONDITION CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER REQUESTING ADDITIONAL DATA
3	ERROR MESSAGE

LEVEL 4
COMPONENTS OF STATE K.B.B
MODIFY OM

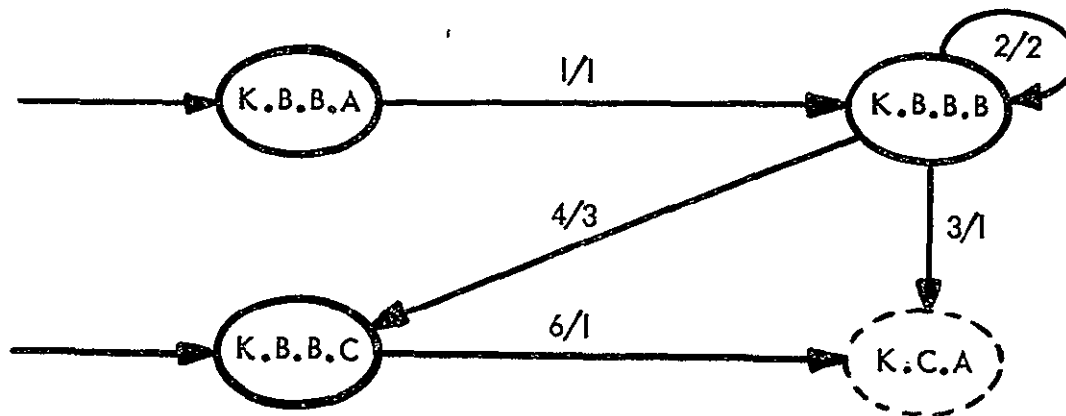
***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
K.B.B.A	RETRIEVE SDD FOR OM
K.B.B.B	PERFORM OM MODIFICATIONS
K.B.B.C	REPORT ERROR

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.B.B.A	K.B.B.B	1	1
K.B.B.B	K.B.B.B	2	2
	K.B.B.C	4	3
→K.B.B.C	→K.C.A	3	1
	→K.C.A	6	1

TRANSITION DIAGRAM



K.B.B : MODIFY OM
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDJ FOR OM RETRIEVED
2	MODIFICATIONS INCOMPLETE
3	MODIFICATIONS COMPLETE
4	ERROR CONDITION ENCOUNTERED DURING MODIFICATION
6	ERROR CONDITION CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER REQUESTING ADDITIONAL DATA
3	ERROR MESSAGE

LEVEL 4
COMPONENTS OF STATE K.B.C
MODIFY JOB

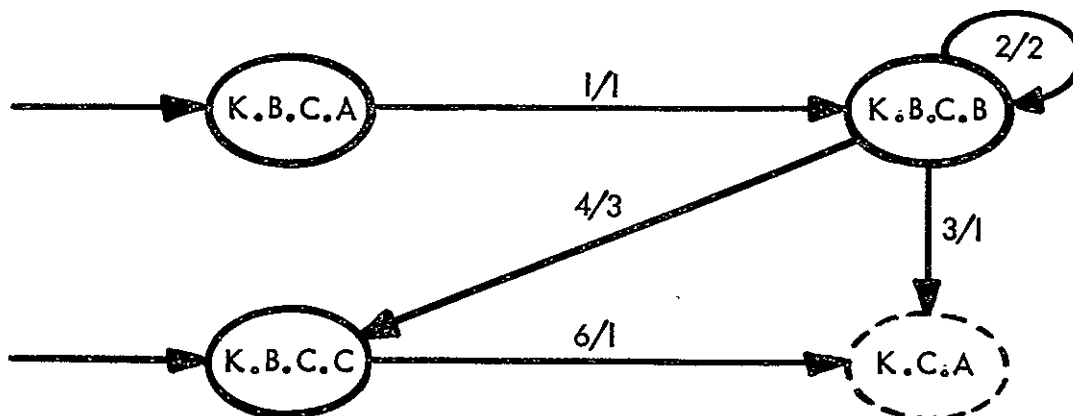
***** STATE DESCRIPTIONS *****

STATE	LONG NAME AND TEXT
K.B.C.A	RETRIEVE SDD FOR JOB
K.B.C.B	PERFORM JOB MODIFICATIONS
K.B.C.C	REPORT ERROR

***** ALLOWED TRANSITIONS *****

FROM STATE (P = ENTRY)	TO STATE (P = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.B.C.A	K.B.C.B	1	1
K.B.C.B	K.B.C.B	2	2
	K.B.C.C	4	3
→K.C.A		3	1
→K.B.C.C	→K.C.A	6	1

TRANSITION DIAGRAM



K.B.C : MODIFY JOB
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDO FOR JOB RETRIEVED
2	MODIFICATIONS INCOMPLETE
3	MODIFICATIONS COMPLETE
4	ERROR CONDITION ENCOUNTERED DURING MODIFICATION
6	ERROR CONDITION CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER REQUESTING ADDITIONAL DATA
3	ERROR MESSAGE

LEVEL 4
COMPONENTS OF STATE K.B.D
MODIFY DS

***** STATE DESCRIPTIONS *****

STATE LONG NAME AND TEXT

K.3.D.A RETRIEVE SDD FOR DATA SET

MODIFICATIONS ARE TO BE MADE TO A USER DEFINED
DATA SET AND THE CORRESPONDING SDD IS RETRIEVED FOR
USE IN THE SUBSEQUENT PROCESSING.

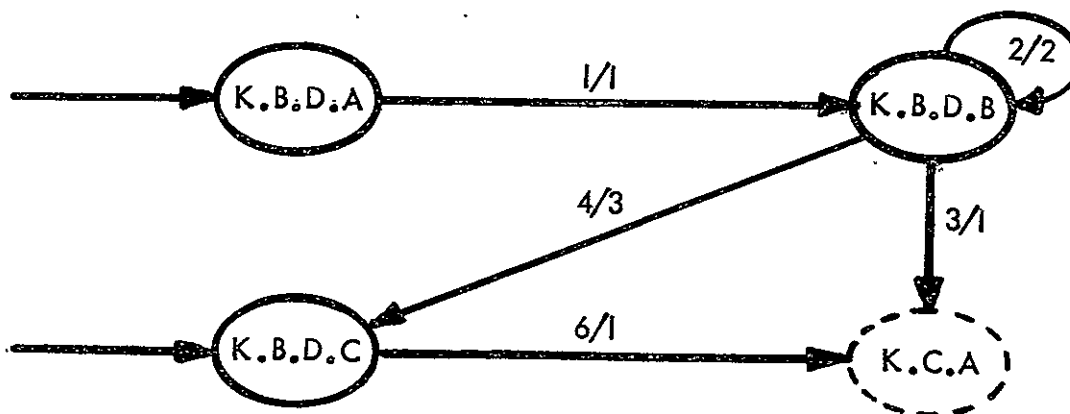
K.3.D.B PERFORM DATA SET MODIFICATIONS

K.3.D.C REPORT ERROR

***** ALLOWED TRANSITIONS *****

FROM STATE (→ = ENTRY)	TO STATE (→ = EXIT)	INPUT / CONDITION	OUTPUT / RESULT
→K.3.D.A	K.B.D.B	1	1
K.3.D.B	K.B.D.B	2	2
	K.B.D.C	4	3
→K.B.D.C	→K.C.A	3	1
	→K.C.A	6	1

TRANSITION DIAGRAM



K.B.D : MODIFY DS
(CONTINUED)

***** INPUT / CONDITION LIST *****

NUMBER	TEXT
1	SDD FOR DS RETRIEVED
2	MODIFICATIONS INCOMPLETE
3	MODIFICATIONS COMPLETE
4	ERROR CONDITION ENCOUNTERED DURING MODIFICATION
6	ERROR CONDITION CLEARED

***** OUTPUT / RESULT LIST *****

NUMBER	TEXT
1	-
2	MESSAGE TO USER REQUESTING ADDITIONAL DATA
3	ERROR MESSAGE

APPENDIX B

DETAILED PROBLEM SOLVING MODEL

B.1 GENERAL WORK FLOW

The design procedures in Volume II are representative of the type and organization of tasks necessary to accomplish the design of an aircraft. However, a computer system designed only to perform the tasks shown in Volume II in the sequences given would have limited value. A generalization of the design procedure into basic elements is needed so that the design of the computing system will support the development of a wide range of air and space craft. The work organization presented in Volume II has been divided into five basic activities. These activities, and their relationship, are shown in figure B.1. Each of the nodes in figure B.1 should be looked upon as actions. The nodes are defined as follows:

- PLAN The determination of objectives and constraints which define a desirable product and the development of a plan of activities to achieve these objectives within the constraints.
- PREPARE Setting up to do work.
- MODIFY Altering preparations to do work when it can be done without changing the plan. Generally, this is due to contingencies which are minor relative to the overall plan.
- WORK The activity which aims directly towards completion of a meaningful step in the plan.
- REPORT Recording and/or making visible the results of WORK and determining if the planned work is done.

The work flow diagram shown in figure B.1. was used to make the following observations:

- a) Entry can be made at any node; however, activity in preceding nodes affecting the entry node must have been completed.
- b) The nodes are coupled; i.e. activity in a node is directly affected by the quality of the information coming to it from a preceding node.

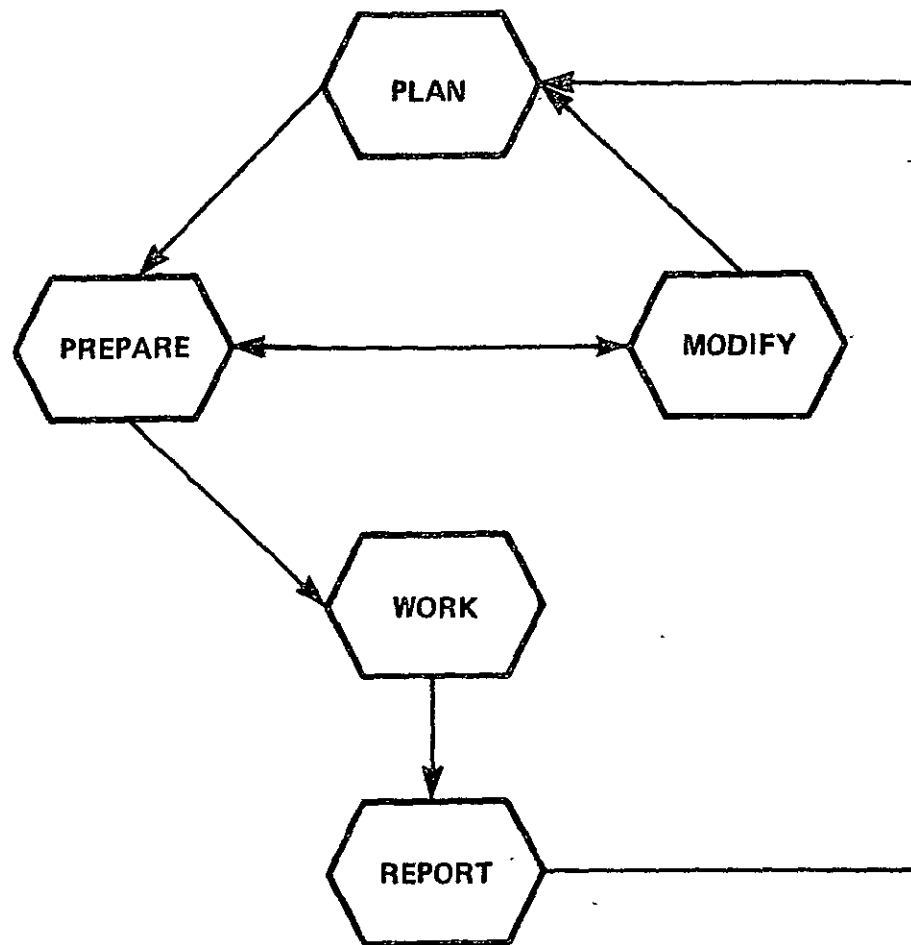


Figure B.1 General Work Flow

- c) All nodes are not connected to all other nodes, nor do all connecting lines have double arrowheads. Examples are:
- 1) WORK always terminates in a REPORT with a return to PLAN. This preserves accountability and management control.
 - 2) There is no direct connection going from PLAN to WORK bypassing PREPARE. Doing so would encourage resource waste and ineffectiveness in WORK.
 - 3) MODIFY always results from an attempt to PREPARE to do WORK. The result of MODIFY is either that a change can be made and PREPARE continue without

affecting PLAN or that an error or weakness has been exposed in PLAN that must be corrected before PREPARE can continue.

- d) The flow diagram can be applied to a single person or a group of persons. Some nonqualitative observations on efficiency can be made. For example, working only within the WORK node is the mark of an undisciplined individual and a poorly managed organization. Working without the REPORT node signifies a lack of personal or supervisory review of work progress relevant to the plan.

These considerations are typical of human planning and organizing functions. The IPAD system will work compatibly within this environment.

A study was also made of how product development is generally organized. It was found that there is a hierarchy of planning and management control as shown in figure B.2. There is a flow of information through the levels of the hierarchy with each level essentially summarizing the level below and providing feedback about the direction of work. The labels of company, product, etc., are somewhat arbitrary. The terms in the parenthesis are basic descriptors of the primary interest at each level. These descriptors are only representative and are not accurate for all organizations. There are, however, several common characteristics.

- a) There is a level at which real work on the product design is accomplished by the user. Above that level, the activity is planning and management control. Below that level, the work is preparation of tools and methods. In the hierarchy shown in figure B.2, real work on the product is at the subtask level.
- b) The user at a particular level tends to transmit information above him and desire action below him.
- c) Those above the user are interested in what is being done, those below the user are interested in how things are done, while the user concentrates on the actual doing, varying his interest between what and how depending on the immediate situation.
- d) The number of levels shown in figure B.2 is not uniquely six, but it is neither large nor small compared to six.

Some further comments on a) are useful. In general terms, there are activities that relate to thinking, planning, and preparing criteria from which to work. In figure B.2, these activities are included in working out the product description, refining the description into a group of tasks that signify areas of responsibility, and further refining each of the tasks into subtasks that signify actual work packages. Once the work package is defined, activity centers around collecting and defining tools and methods into packages with which to perform the work. This latter activity is called a "job" in figure B.2. If the tools or methods do not exist or have to be modified, another class of activity is required called an "activity" in figure B.2. From these considerations, the IPAD system should be formulated around a principal work activity called a subtask and that all other activities support or relate to working on a subtask.

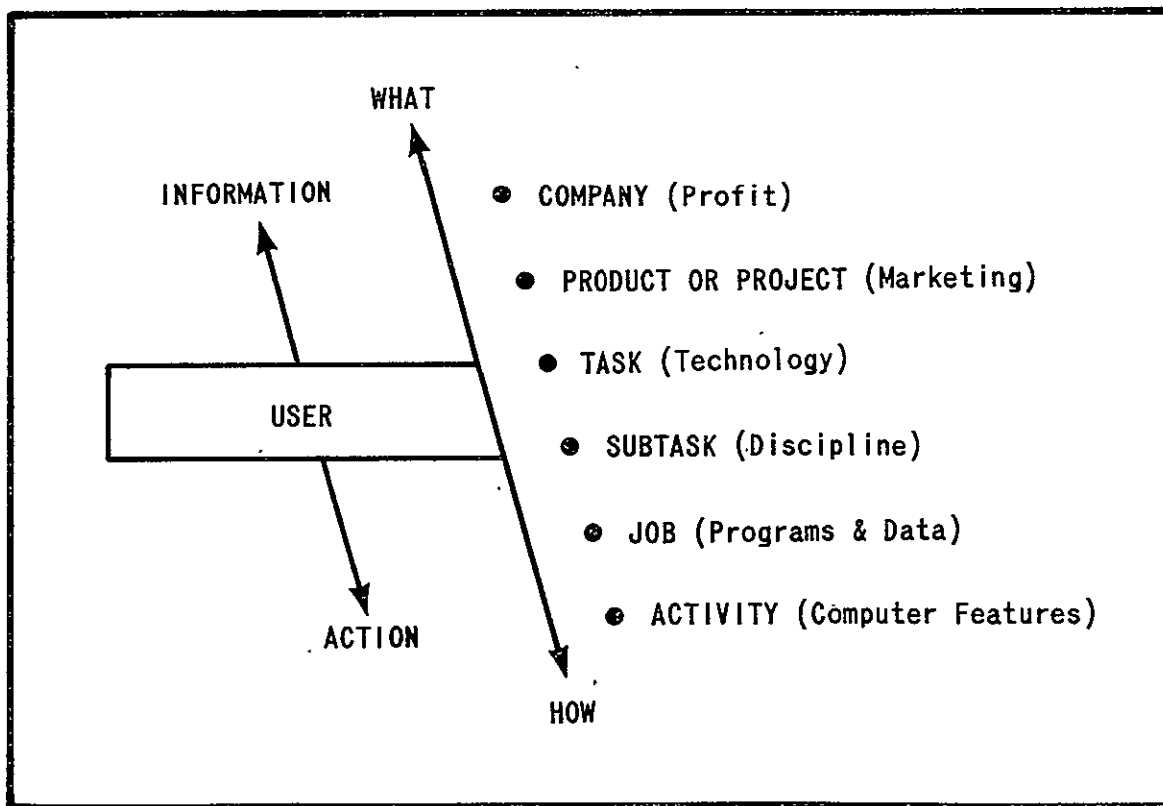


Figure B.2 Organizational Hierarchy of Product Design

Figures B.3 through B.7 are an expansion, of each node of figure B.1. For example, figure B.3 is an expansion of PLAN with the unexpanded nodes, MODIFY, PREPARE, WORK and REPORT, appended to show connectivity.

The following definitions will be helpful in reading the flowcharts and descriptions:

- Objective - The result to be achieved from a design activity.
- Constraint - A bound placed on a design activity as a limit or assigned value.
- Report - The collection of objectives and constraints which, when satisfied, will describe and specify the product design and serve as a basis for judgement on the success or quality of intermediate design activities. (This is distinct from the verb "REPORT" used as one of the nodes of the work flow model.)

Other definitions are given in section 4.0 of volume IV.

B.2 "PLAN" NODE DEFINITIONS - Figure B.3

Node 1 - DEFINE Product into Tasks

DEFINE is a general node meaning refinement of an existing definition by dividing it into subparts; for example, in this case, dividing a product development program into several tasks. Hence, this define is a description of the desired product objectives and all the major work tasks necessary to achieve these objectives. The results of this definition need to be available for review, revision, and comparison throughout the product development period. This, and other similar information generated in later DEFINE blocks, is compiled in a record which serves as a continuing source of direction.

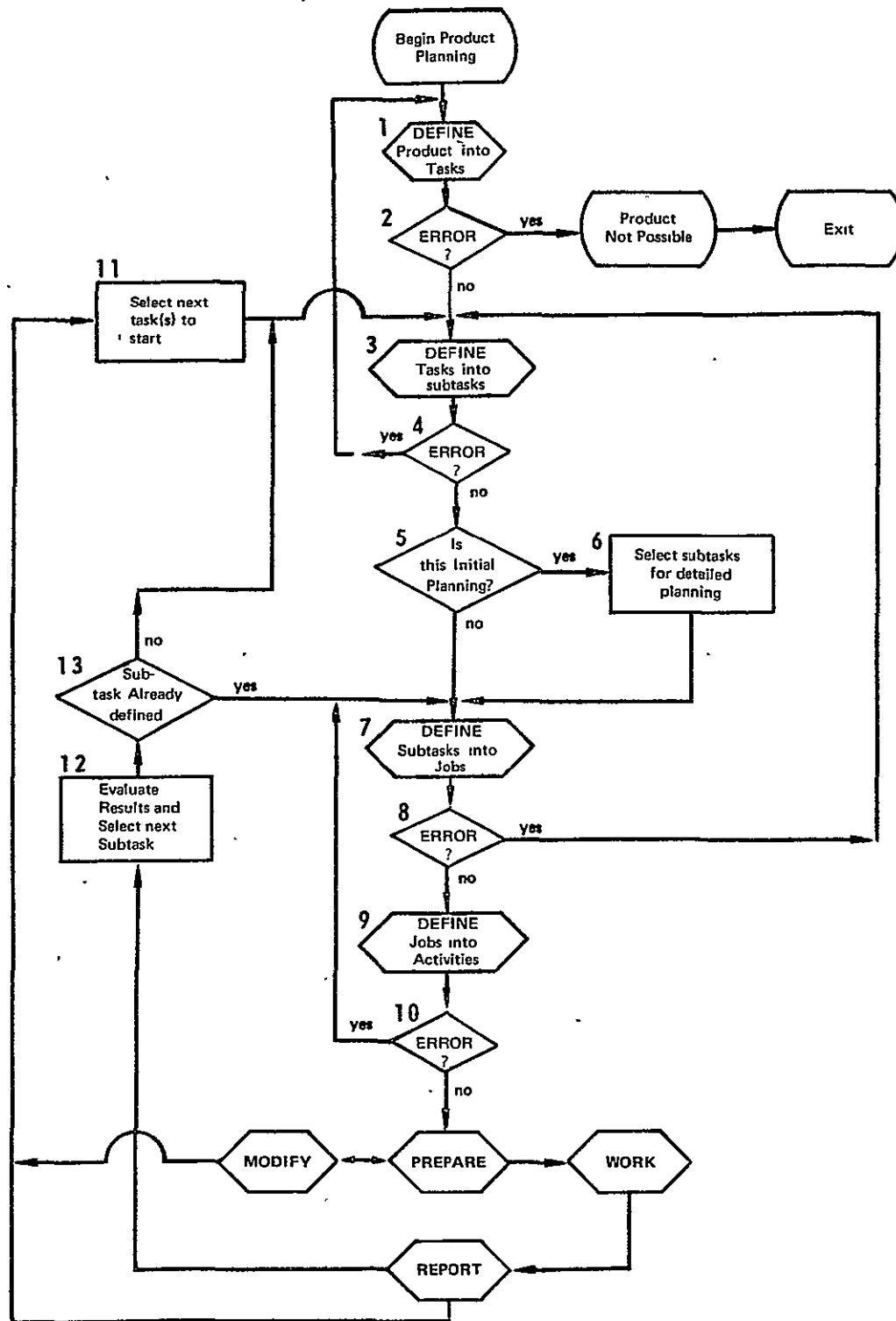


Figure B.3 An Expansion of PLAN

- Node 2 - ERROR?
A YES choice in an error branch means an inconsistency was found during refinement in the previous DEFINE that requires reconsideration of a higher level DEFINE. In practice, the ERROR box at Node 2 probably does not exist because the implications of a YES choice at this level are too far reaching. Any changes in the original product definitions would have been corrected prior to compiling the record in Node 1.
- Node 3 - DEFINE Tasks into Subtasks
See Node 1 definition. Tasks are divided into subtasks. Note that an error found in Node 8 may force a redefinition in Node 3. This is primarily a dependent variable problem that would not exist if the tasks were independent. Subtask definitions are characterized by a record of objectives and constraints. A subtask appears to be the basic level at which meaningful work on the product is done. The objective in PLAN is to identify the necessary subtasks closely enough so that scheduling and resource requirements can be estimated and to assure that a workable set of activities has been defined.
- Node 4 - ERROR?
See Node 2 definition.
- Node 5 - Initial Planning?
The dividing line between planning and actual work on the product design is at the subtask level. There may be a division in the PLAN activity centered around this distinction. During initial planning, some critical subtasks may be planned in detail, i.e., while others may be left until the work actually begins.
- Node 6 - Select First Subtask(s)
There will be many subtasks. Critical ones may be selected for initial planning, others left for later.
- Node 7 - DEFINE Subtasks into Jobs
The relationship between the subtask definition and the tools and methods available to do the work is established here. Within IPAD, this activity usually means collecting and assembling computer programs into working packages. See Node 1 definition also.
- Node 8 - ERROR?
See Node 2 definition.

- Node 9 - DEFINE Jobs into Activities
Specific computing techniques enter into the planning at this level. Any job involving computational tools already developed and tested need not enter into this level of planning. Typically, the non-computing user would only work in this level during the original tool planning and specification phase. This is the typical level at which computer programming and system design support will be utilized.
- Node 10 - ERROR?
See Node 2 definition.
- Node 11 - Select Next Task
This is either
- a) a return from REPORT at the completion of a task with the intent of beginning a new task, or
 - b) a return from MODIFY with the conclusion a subtask is unworkable and must be redefined at the task level.
- Node 12 - Evaluate and Select Next Subtask
This is a return from REPORT after subtask completion and before task completion. The completed results of this subtask are evaluated against the task plan to determine if the results of the subtask are satisfactory and what the next subtask is.
- Node 13 - Subtask Already Defined
If the review at Node 12 introduces an unplanned subtask, a redefinition at the task level is required.

B.3 "PREPARE" NODE DEFINITIONS - Figure B.4

- Node 1 - Sit Down
"Sit Down" is the activity of addressing ones self to performing a job. It expresses the act of focusing attention to a relatively narrow field of potential accomplishment that can be completed in hours or days as opposed to weeks or months. That is, the time span for completion is small compared to the total task and very small compared to the product. Generally, the job is capable of being accomplished through the primary involvement of one user.

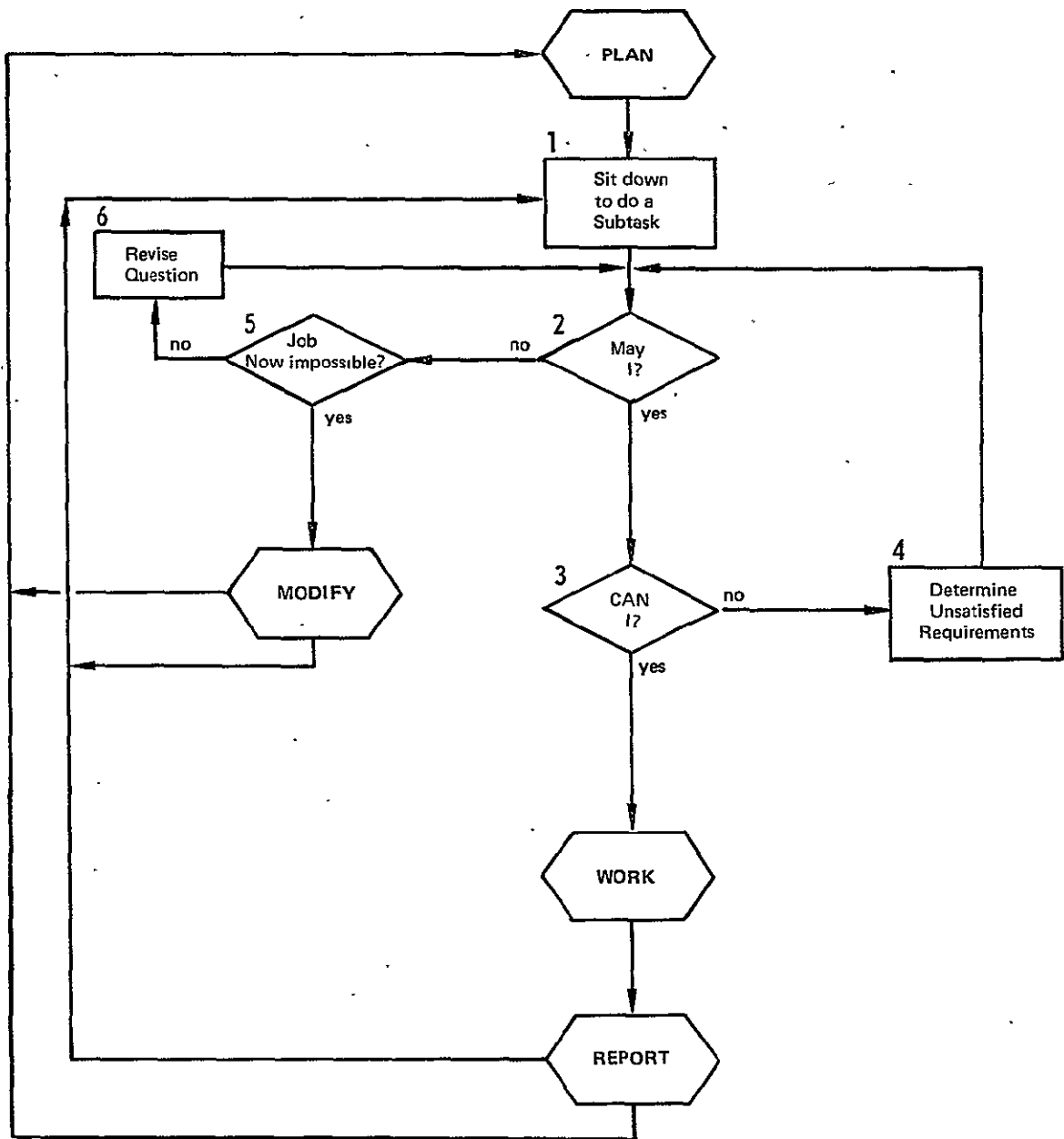


Figure B.4 An Expansion of PREPARE

Node 2 - May I?

Given the user has some plan of action, he then proceeds to initialize. Initialization generally consists of collecting all the data and capability the user understands to be pre-requisites to starting. The user knows in principle what data, computer programs, sequences, etc., is needed. He must now package specific items and in so doing determines availability, adequacy, permission requirements, etc.

Node 3 - Can I?

While "May I?" concentrates on availability, general adequacy, and authorization; "Can I?" emphasizes workability. That is, "Will the program execute on my hardware?"; "Will sequential computer programs interface data automatically?"; etc. If the user has complete control over his tools, this step diminishes in importance. When the user is utilizing tools developed and controlled elsewhere, this step can be of commanding importance.

Node 4 - Determine Unsatisfied Requirements

Any requirements not previously identified must be satisfied before continuing.

Node 5 - Job Impossible?

The job may be threatened by any of the following:

- a) Requesting access without permission.
- b) Requesting access to nonexistent information.
- c) Illogical relationships in planning not identified until now.

The problem is resolved by either taking an alternate equivalent route or by modifying the plan.

Node 6 - Revise Question

Local variations in PLAN implementation are possible. This box allows for such a variation (provided it does not change the basic plan), as well as for the conclusion that the user asked a bad question.

B.4 "MODIFY" NODE DEFINITIONS - Figure B.5

Node 1 - Can I Redefine Job?

The current job is not executable and somehow must be altered if the subtask is to be continued without compromise. Sufficient information is required to avoid an unnecessary NO answer to this question, and be clear about why the job is impossible in its current state. A self teaching system would be useful here.

Node 2 - Redefine Job

The full power of job construction and modification must be available at this point.

Node 3 - Can I Redefine Subtask?

Because the current job execution is not possible, the subtask definition is threatened. A NO answer to this question means a review of basic plans.

Node 4 - Redefine Subtask

The remarks under Node 2 apply here. Redefining the subtask may or may not require job redefinition.

B.5 "WORK" NODE DESCRIPTIONS - Figure B.6

Node 1 - Execute

Activities in this box tend to be associated with computation in the general sense. Upon completion of this box, the user expects results from which subtask or task decisions will be made. To the scientific user this is where the "floating point arithmetic" takes place. The distinction between some activities done in WORK and those done in PREPARE can be user dependent. For example, FORTRAN compilation could be done in either place. This node is where the primary use of the central processing unit occurs. All technical code will execute in this box.

Node 2 - Interaction Required?

The user may need to interact with an executing computer program to query certain parameters, alter sequences, stop execution, test optimization convergence, etc. Any type of control that is not exercised through the technical code falls into the category of system code interaction.

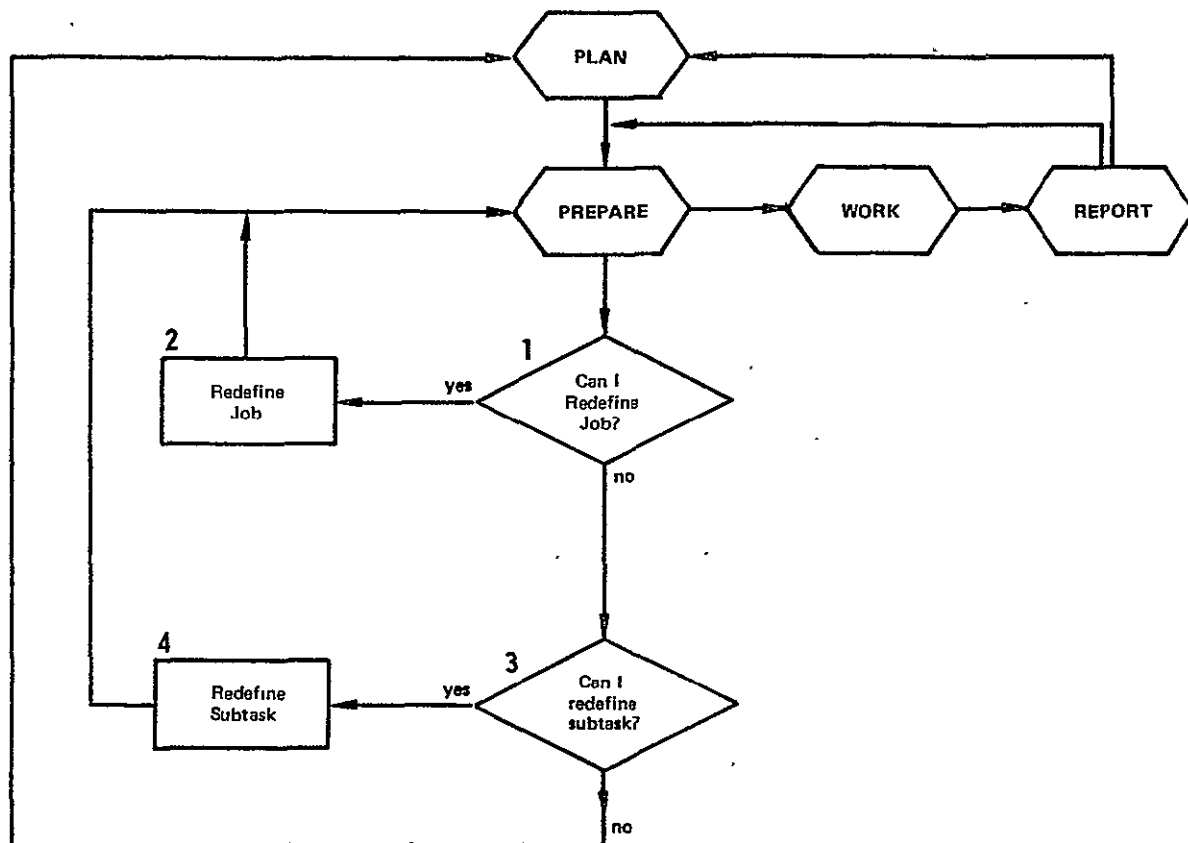


Figure B.5 An Expansion of MODIFY

ORIGINAL PAGE IS
OF POOR QUALITY

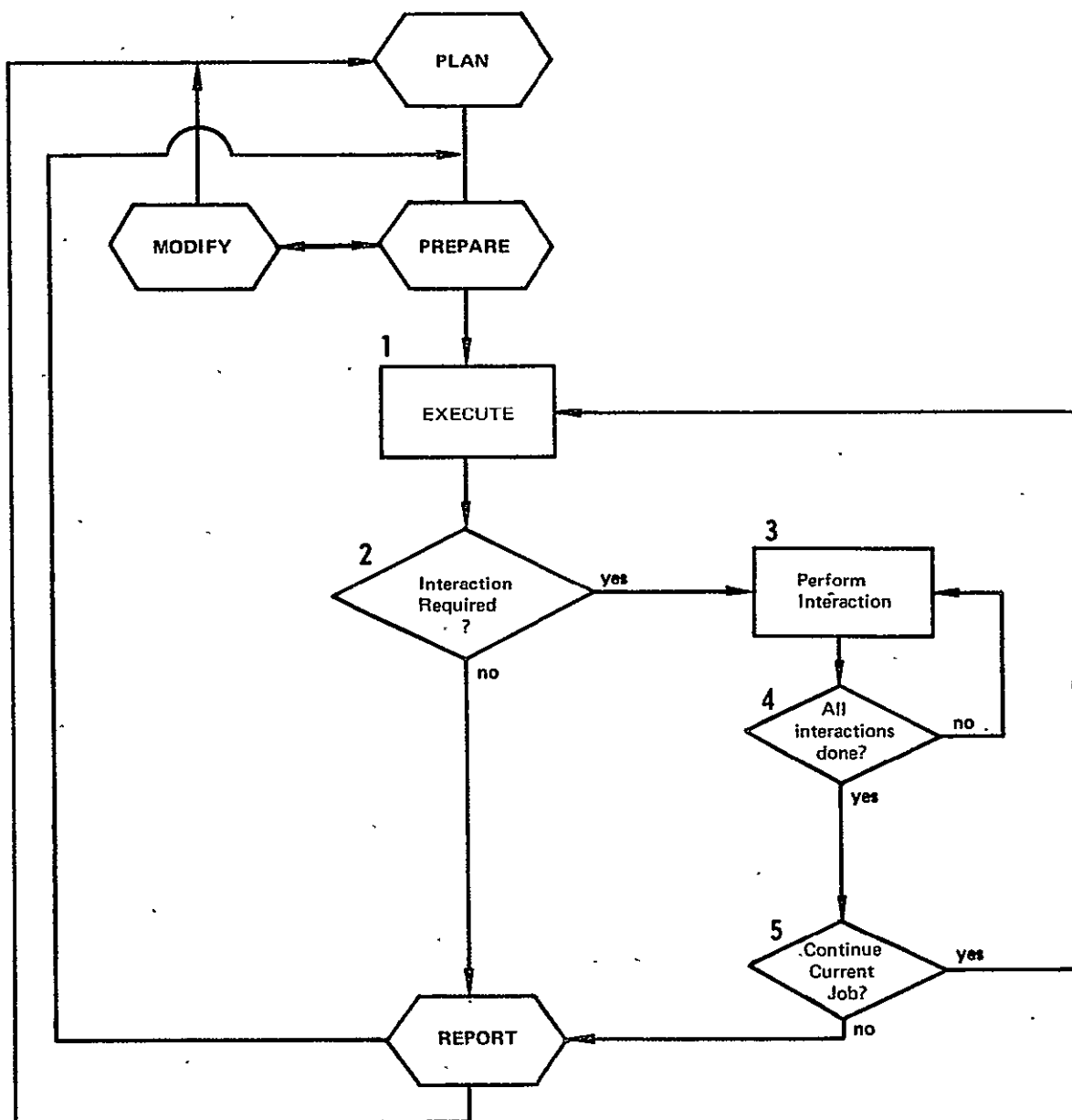


Figure B.6 An Expansion of WORK

- Node 3 - Perform Interaction
In this activity, the user inserts his judgment and control into the execution sequence. Activity status needs to be maintained while interaction takes place. The point at which interaction will take place is specified at the time of interaction without having to alter the initial execution plan. The physical and logical interface characteristics are all important and must support any aim the user may have in desiring interaction.
- Node 4 - All Interactions Complete?
- Node 5 - Continue Current Job?
When interactions are complete, the current job may or may not be useful. If yes, restart must be able to take place. If no, the option must be available to continue the job at a later time.

B.6 "REPORT" NODE DESCRIPTIONS - Figure B.7

- Node 1 - Save Job and Subtask Data
Since the current job is not to be continued at the moment, restart data may need to be retained for restart at a later time. Whatever has been completed in this job needs to be logged on the subtask record sheet unless the user desires an elimination of all job effects and accomplishments. All data saved must be sufficiently labeled to avoid ambiguity and confusion later on. Time and date are minimal components in this identification.
- Node 2 - Save Task Data
This implies a knowledge of task data as opposed to subtask or job, which implies a knowledge that a task exists. Thus some kind of task plan must be visible in the system, or at least identifiers of task level data that can be matched must exist. This can be extremely important when data from one subtask feeds another subtask. Otherwise a user could invoke a job where execution is dependent upon data sets not yet generated.

ORIGINAL PAGE 15
OF POOR QUALITY

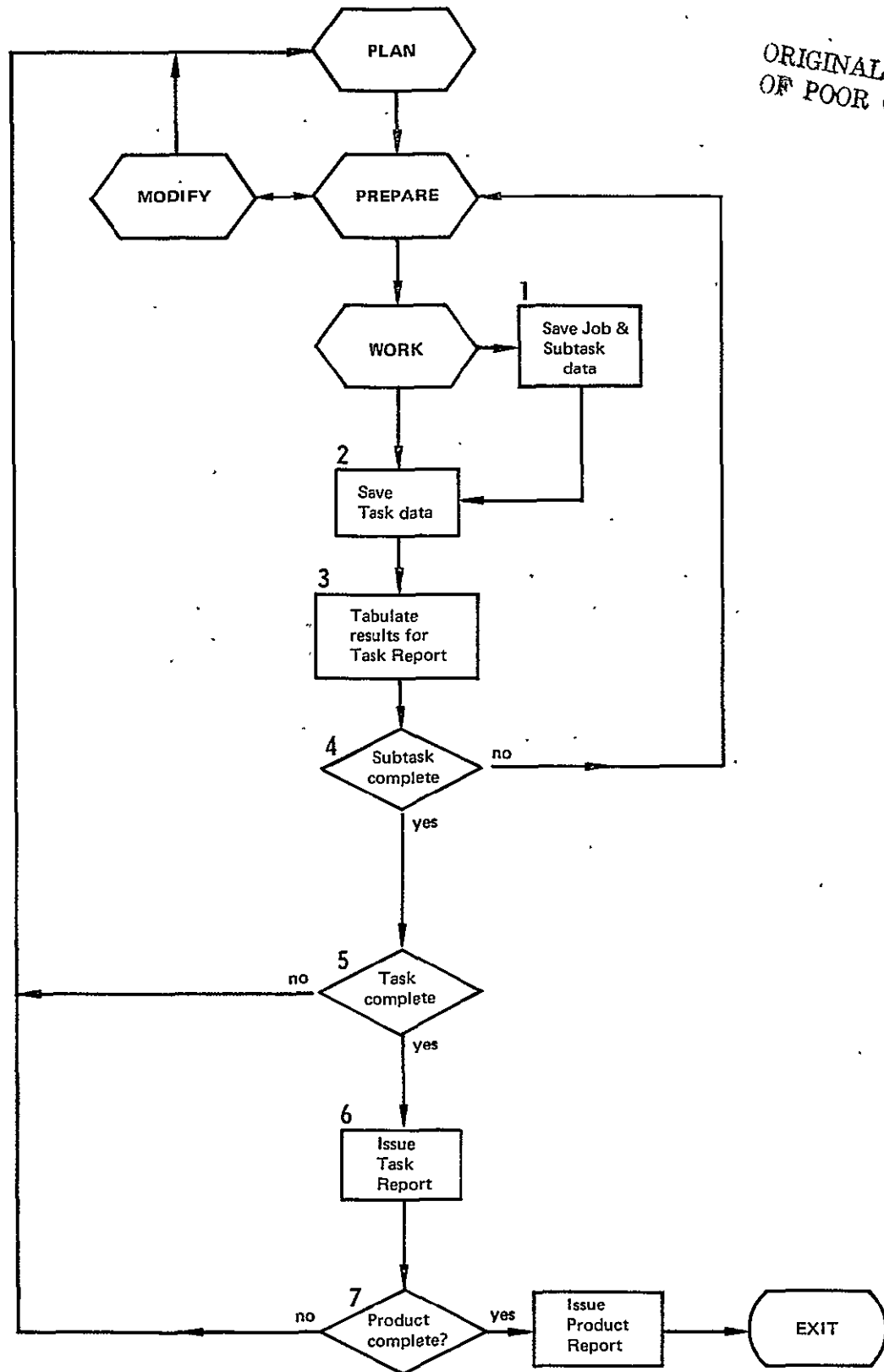


Figure B.7 An Expansion of REPORT

- Node 3 - Tabulate Results for Task Report
Assuming that the report is defined and carried over from PLAN, any subtask data that is also part of the task report should be identified and logged so that the degree of report completion is clearly observable on a timely basis.
- Node 4 - Subtask Complete?
A subtask plan must be available (even if it is only a mental picture) before this question can be readily answered. A NO answer implies that the user is now in the middle of an interrupted subtask, trying to follow a path to allow restart of the subtask.
- Node 5 - Task Complete?
Similar to Node 4.
- Node 6 - Issue Task Report
Report generation capability is clearly needed here. Each item in the report needs to be threaded to the data which contributed to the item. The report should have several levels of detail, depending upon the level within figure B.2 to which it is issued.
- Node 7 - Product Complete?
Similar to Node 4.
- Node 8 - Issue Product Report
Similar to Node 6.

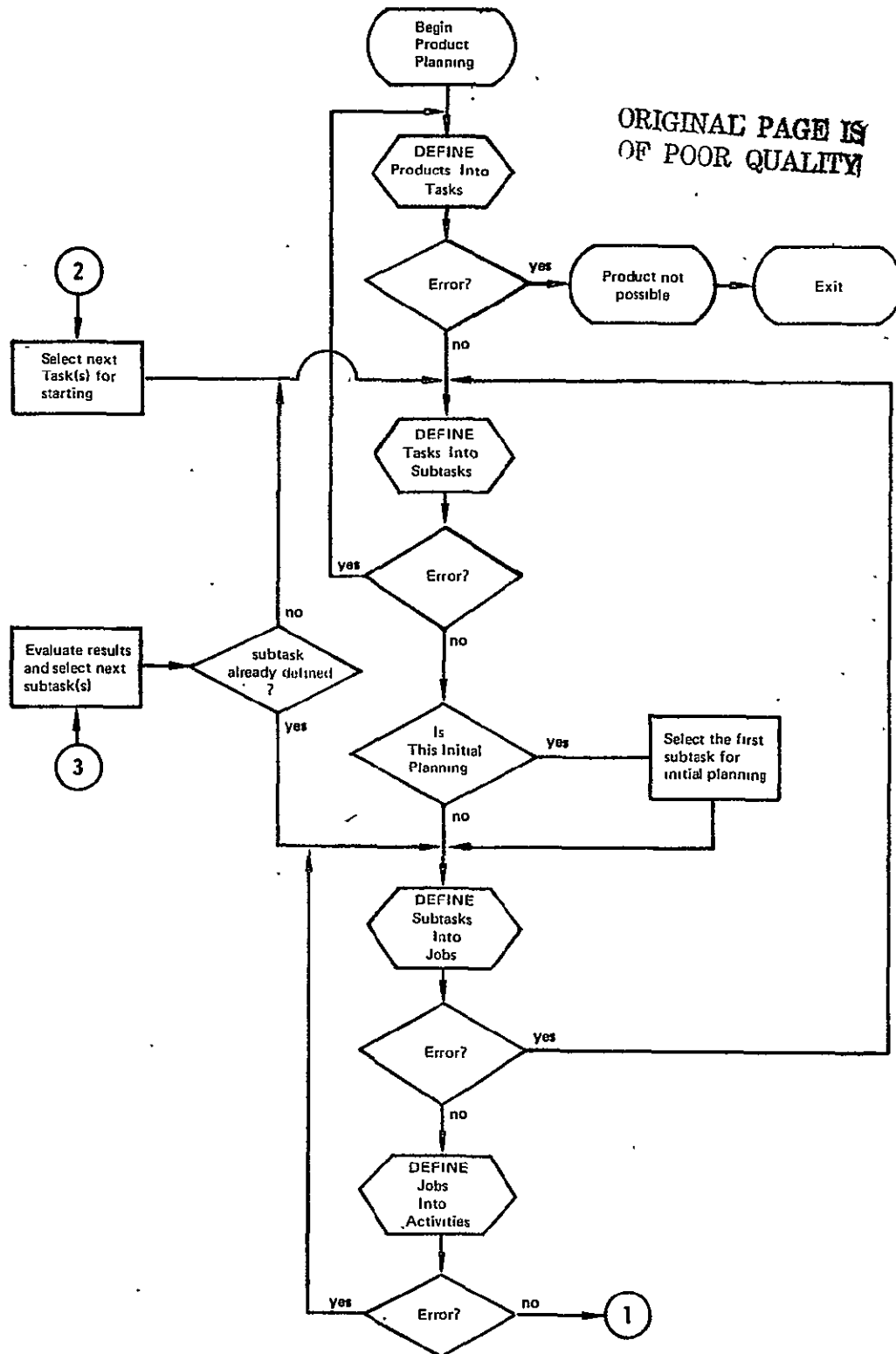


Figure B.8.1 An Expansion of the Total Work Flow Model

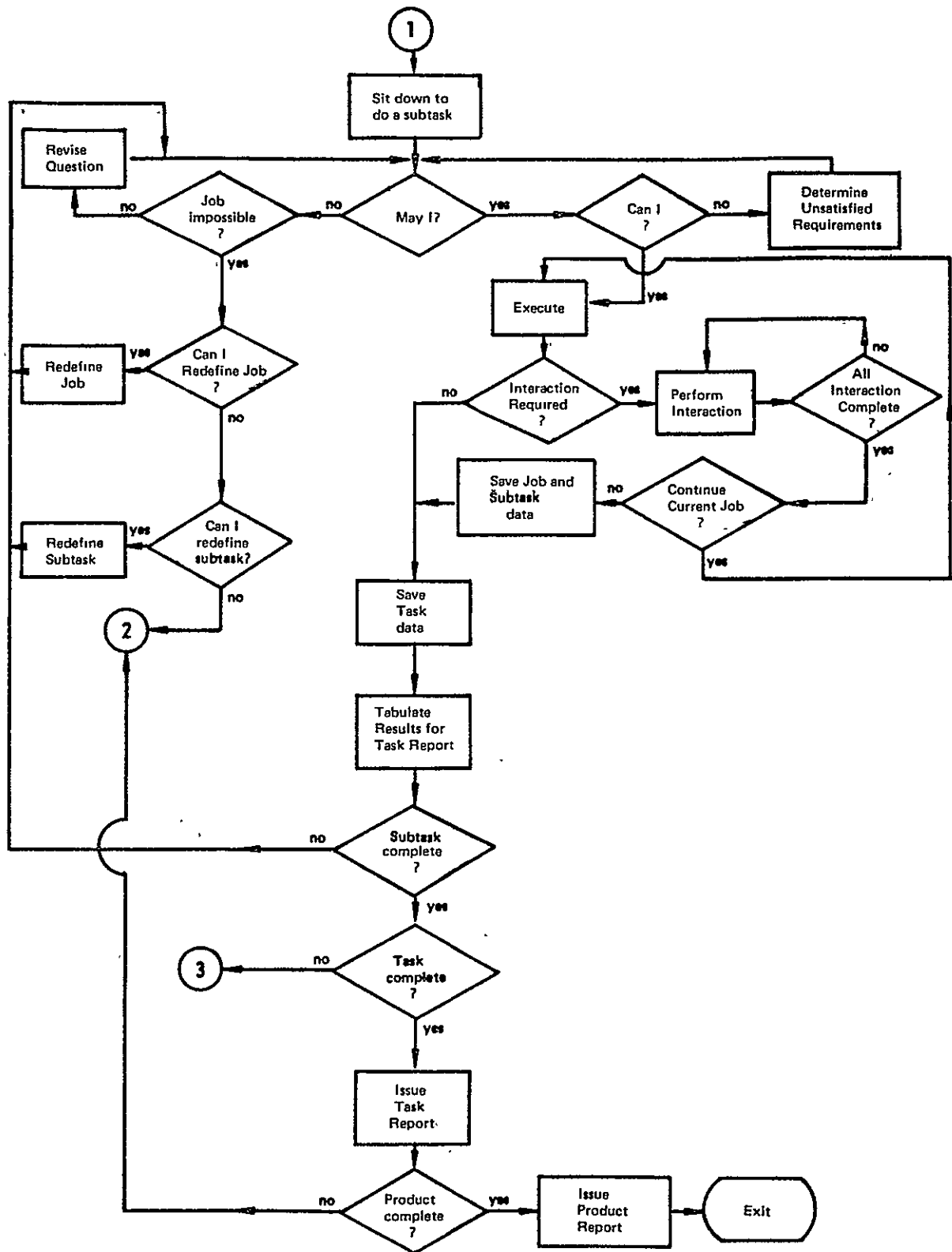


Figure B.8.2 An expansion of the Total Work Flow Model (Cont'd.)

APPENDIX C

MIGRATION OF IPAD SOFTWARE

A special study was conducted by the Control Data Corporation as a subcontract of the Boeing IPAD contract, to investigate the problems of:

- a) migrating applications programs and the IPAD system software from a 3rd generation computer to a 4th generation computer within a computer family and
- b) migrating applications programs and IPAD system software across 3rd generation computer families.

The following is the final report of their study.

ORIGINAL PAGE IS
OF POOR QUALITY

IPAD
SOFTWARE MIGRATION

5 January 1973

Approved: J. A. Kershaw
J. A. Kershaw
General Manager

Submitted by
W. E. Glass
Control Data Corporation
Advanced Systems Laboratory
4201 North Lexington Avenue
St. Paul, Minnesota 55112

TABLE OF CONTENTS
IPAD SOFTWARE MIGRATION

ORIGINAL PAGE IS
OF POOR QUALITY

SUMMARY AND CONCLUSIONS

1.0 THEORETICAL BACKBROUND

- 1.1 The General Problem
- 1.2 Language Conversion
- 1.3 A Model Language Converter
- 1.4 Problems of Language Conversion
- 1.5 Comparison of Conversion Methods
- 1.6 Implementation of Language Converters
- 1.7 Preliminary Remarks on Common Language Design

2.0 FORTRAN SOURCE CODE TRANSLATION ON THIRD GENERATION COMPUTERS

- 2.1 Features in CDCF not in IBMF
- 2.2 Features in IBMF not in CDCF
- 2.3 Syntax Differences Between CDCF and IBMF
- 2.4 Machine Dependent Statements in CDCF and IBMF
- 2.5 Input/Output and Data Transfer Statements in CDCF and IBMF
- 2.6 Implementation Restrictions in CDCF and IBMF
- 2.7 CDCF and IBMF Interfaces with Job Control Languages
- 2.8 Translation from IBMF to CDCF

3.0 INTRODUCTION TO THE DEVELOPMENT OF A MACHINE INDEPENDENT FORTRAN

- 3.1 Preliminary Remarks
- 3.2 IPAD FORTRAN (IPADF)
- 3.3 FORTRAN Dialect to IPAF Translation
- 3.4 IPAD Implementation Language

4.0 MIGRATION OF OM'S FROM THIRD GENERATION TO FOURTH
GENERATION COMPUTERS

4.1 Introduction

4.2 IPADF Extended for Vector and String Processing (IPADFV)

4.3 IPADF to IPADFV Translation

SUMMARY AND CONCLUSIONS

A solution of the general IPAD problem (Section 1.1) includes, as one of its principal parts, development of methods for moving operational modules (OM's) freely among the various computers in the IPAD system. Since most of the OM's are written in some form of FORTRAN, a solution for this case alone can be expected to be useful.

A number of methods for solving this migration problem, at least in principle, are examined in Section 1.2. Of these, only two showed sufficient promise to be retained for further consideration. The first, Method 1, requires that each source language program be translated to a common language before compilation and execution. The second, Method 3 in the original list, is based on a pairwise set of source - host translators. The principal advantage of Method 1 is that the common language becomes, by definition, the IPAD standard. The advantage of Method 3 is that its initial cost is probably lower, but as new dialects enter the system, new translators must be written, and documentation standards would be difficult to enforce. Method 1, thus appears, on balance, to be the preferred choice, and is accordingly recommended.

The design of the common IPAD language is the next concern. Three requirements are basic. It must be possible to translate existing programs to it, the translated programs must not contain machine dependent code, and the language must be extensible to accommodate fourth generation computers entering the IPAD system.

As a preliminary step, two FORTRAN dialects, one for CDC computers and the other for IBM computers, are examined in detail and differences in syntax and usage noted (Section 2). Syntax differences are numerous, but do not constitute a serious translation problem, since the intended interpretation for the source code is known, and the corresponding statements in the host language can be constructed from this knowledge. Several examples of this syntax conversion are given in Section 2.8.

A more difficult problem arises out of the interaction among the program, the job control language, and the compiler. Fortunately, the areas in which this problem are most likely to arise are known, and even if translation cannot be

carried out automatically, the suspect parts of the program can be flagged for programmer examination.

The most difficult translation problem occurs when code adhering to a common FORTRAN syntax makes use of machine dependent constants or variable values, since here there may be no indication that the code is machine dependent, and therefore, there is nothing for the translator to detect. Even here, however, there are constructions in which this problem is more likely to occur, and these too can be flagged for review. The conclusion that follows is that much of the translation process can be automated, but a substantial residue remains for hand translation, and the significant problems are the detection of machine independent code and then fathoming of the programmer's intent.

It follows from examination of FORTRAN dialects in current use that the common IPAD language can be machine independent only if it requires that much detail that is now implicit in the computer environment for which the program was written be specified explicitly. Accordingly in the proposed common language, IPADF (Section 3), it is required that all variables be declared, either explicitly or by class, together with their lengths in appropriate units. The collating sequence must either be given explicitly, or fixed once and for all. (The decision here was left open, pending further study). Explicit reference to overlays is banned, but it may be desirable to allow declaration of variables by level in a presumed hierarchy of storage.

To allow for extension of the language to the new class of vector and string processors IPADF allows elementary arithmetic and logical operations on one dimensional arrays, but not on subsets of declared arrays. This rationale allows operations favoring fourth generation computers that can still be implemented readily on third generation machines.

More extensive vector and string operations analogous to those available as primitive functions in APL are reserved for IPADFV, the fourth generation IPAD FORTRAN, which contains IPADF as a subset. The architectural differences between third and fourth generation machines is so fundamental that it can be assumed that most OM's will in time be reprogrammed for the newer computers. Consequently, IPADFV is not expected to be introduced until fourth

generation machines have effectively replaced their predecessors. An example illustrating this reprogramming is given in Section 4.3.

The conclusion reached in this study is that the IPAD software migration problem is best solved by translating current OM's into, and writing all future OM's in a common machine independent FORTRAN based language, IPADF, developed especially for IPAD, that can be extended to include vector and string processing in its fourth generation version, IPADFV.

IPAD SOFTWARE MIGRATION

1.0 THEORETICAL BACKGROUND

1.1 The General Problem

The IPAD software migration problem can be stated as follows: given an open-ended set of programs, called Operational Modules (OMs), written in several languages for different computers, design a machine independent system, IPAD, in which the OMs are linked by an executive program through a data management subsystem to a common data bank. The system should execute as a job in batch mode under the standard operating system at any installation having the minimum equipment configuration required.

1.2 Language Conversion

Suppose that m different source languages are represented among the OMs and that IPAD must execute on n different host computers. If d of the source languages are also languages for the host computers, then $mn-d$ source language to host machine language conversion algorithms will be required.

Most scientific programs (in the United States at least) are written in some form of FORTRAN. It will be assumed, therefore, that the OMs are all written in a dialect of FORTRAN and that each source and host computer is provided with a compiler for converting programs to its own machine code.

Now let -

L_i be the i^{th} FORTRAN dialect

M_j be the j^{th} machine language

$P(A, L_i)$ be a program for algorithm A expressed in L_i

$P(A, M_j)$ be a program for algorithm A expressed in M_j

T_{ij} be a translator for converting programs from L_i to L_j

C_{ij} be a compiler for converting programs from L_i to M_j

Also let $A \rightarrow \boxed{B} \rightarrow C$ denote the execution of program B with input A and output C .

With this notation, a software migration process can be described concisely. For example -

$$P(A, L_i) \longrightarrow \boxed{P(T_{ij}, M_r)} \longrightarrow P(B, L_j) \longrightarrow \boxed{P(C_{jk}, M_s)} \longrightarrow P(D, M_k)$$

In words, algorithm A written in L_i is translated to L_j in machine r to produce a program for the modified algorithm B. The new program is compiled on machine s yielding a machine language program for yet another modification of the algorithm which can be executed on machine k. In real processes, r and s normally belong to the set (j, k), but it is not necessary. Nor is it necessary that A, B, and D be the same algorithm. Indeed, there is no general way to decide if they are or not, except at the machine language level where the outputs of a program and its translation can be compared. In what follows, however, it will be assumed that the algorithm is carried unchanged through the steps of a migration process, unless deliberately altered.

The migration process sketched in the example above can be partitioned in three different ways, depending on whether $j = o, i$, or k .

Method 1. $j = o$. For this case, each source language program is translated to a common language, L_o , before compilation. Schematically -

$$P(A, L_i) \longrightarrow \boxed{P(T_{io}, M_r)} \longrightarrow P(A, L_o) \longrightarrow \boxed{P(C_{ok}, M_s)} \longrightarrow P(A, M_k)$$

Here, m translators T_{io} and n compilers C_{ok} are required.

Method 2. $j = i$. Since T_{ii} is equivalent to the identity translation, this method simplifies to -

$$P(A, L_i) \longrightarrow \boxed{P(C_{ik}, M_k)} \longrightarrow P(A, M_k)$$

and requires mn-d compilers.

Method 3. $j = k$. Here -

$$P(A, L_i) \longrightarrow \boxed{P(T_{ik}, M_r)} \longrightarrow P(A, L_k) \longrightarrow \boxed{P(C_{kk}, M_k)} \longrightarrow P(A, M_k)$$

This method requires mn-d translators, but makes use of the already existing host language compilers.

If t represents the cost of a translator, and c the cost of a compiler, then $mt + nc$, $(mn-d)c$, and $(mn-d)t$ are the cost functions for these three methods respectively. The relative ranking of these cost functions depend, of course, on the values assigned the parameters. However, under the plausible assumptions that (i) $d=0$, (ii) $m + n < mn$, and (iii) $t < c$, it is easy to see that Method 2 is more costly than either Method 1 or Method 3. Accordingly, it will be dropped, at least for the time being, and Methods 1 and 3 retained for further consideration.

One other method merits introduction at this time. Let I_{ij} be an interpreter for converting programs from M_i to M_j . Then -

Method 4.

$$P(A, L_i) \rightarrow \boxed{P(C_{ii}, M_i)} \rightarrow P(A, M_i) \rightarrow \boxed{P(I_{ij}, M_j)} \rightarrow P(A, M_j)$$

This method requires mn-d interpreters.

Sometimes translation and execution are combined, so that each instruction in M_i is translated as needed, yielding -

Method 4a.

$$P(A, M_i) \rightarrow \boxed{P(I_{ij}, M_j)} \rightarrow \boxed{P(A, M_j)} \rightarrow R$$

$I \rightarrow$

1.3 A Model Language Converter

It will be useful to have at hand an idealized model of a converter which can represent, in turn, a translator, compiler, or interpreter. Suppose the conversion to be performed by a finite state machine defined by the two functions

$$Y(i + 1) = \Psi \left[S(i), x(i), Y(i) \right]$$

and

$$S(i + 1) = \Phi \left[S(i), x(i), Y(i + 1) \right]$$

where $x(i)$ is the current character from the input source language text, $Y(i)$ is the current (possibly empty) sequence of output characters of the converted text, and $S(i)$ is a vector defining the current state of the machine. The converter, thus, can be conceived as a black box that accepts the source text, one character at a time, and after a certain number have been received, depending on the state, issues a string of one or more characters of the converted text. For certain combinations of input and machine state, conversion may not be possible, and $Y(i + 1)$ will be issued as an error message. In the simplest case, well-defined (as to beginning and end) substrings of input text are replaced by substrings of output text, where a one-one correspondence exists between the two strings. In the worst case, the complete input text must be received by the converter before the first character of the converted text is emitted.

In general, conversions will range between these extremes. Well-defined substrings (called here sentences) are presented to the converter in sequence. As each sentence is received, it will have 0, 1, or many conversions.

If the input sentence has no conversion, and is a syntactically correct sentence of the input language, it denotes an action of the source computer that cannot be expressed in the output language, either (i) because of a shortcoming in that language, or (ii) because no comparable action exists for the host machine. In either case, a special message is inserted in the output text.

If an input sentence has just one conversion, the corresponding output sentence (or sentences) is issued.

If an output sentence has more than one translation, it must be saved until sufficient text is accrued to resolve the ambiguity.

1.4 Problems of Language Conversion

There are a number of problems associated with the design of a practical converter. The main task is to establish a correspondence between the sentences of the source and host languages. Most sentences in most languages are ambiguous when taken by themselves. In FORTRAN, statements (sentences) are made up of a fixed part and a variable part (which may be empty). The

fixed part serves to identify the class to which the statement belongs, and, in fact, is a sort of distributed name for that class. If the variable part is absent, the statement cannot be ambiguous; it will either have exactly one meaning, or none. On the other hand, the variable part, when present, is composed of parameters whose values in a given program may be determined by the characteristics of the source computer; i. e., the statement may be machine dependent. If every statement containing parameters needs to be analyzed in its (often unbounded) context to ascertain its true meaning, the conversion task, if not impossible, becomes one of truly formidable proportions.

Despite the fact that many FORTRAN statements are potentially ambiguous, most statements actually appearing in a pair of FORTRAN programs written in different dialects are identical in form and have almost the same meaning. Of the remainder, most involve merely syntax differences, in which again essentially the same meaning is expressed in another way. The residue, comprising only a small part of the source program, requires the most effort, and here the difficulty lies more often in detection of the anomaly than in its correction.

While similar statements in the source language generally have close to the same meaning as in the host language, they are seldom identical, due to differences in the computer's word lengths, data representation methods, memory organization, etc. When a statement has a different meaning for the host machine than for the source machine, a decision must be taken as to which meaning the converted text is to carry. If the host computer's interpretation is accepted, the statement and its conversion are identical. If the source language interpretation is to be preserved, the converted text becomes a set of directions for reproducing the source machine action on the host computer. In the latter case, conversion reduces to imitation (emulation or simulation) of the source computer on the host machine. This is essentially Method 4 above. The advantage is that conversion is exact. The disadvantages are that (i) writing an interpreter for one computer in the language of another is not a simple task, and (ii) simulation of one machine on another leads to very inefficient program execution.

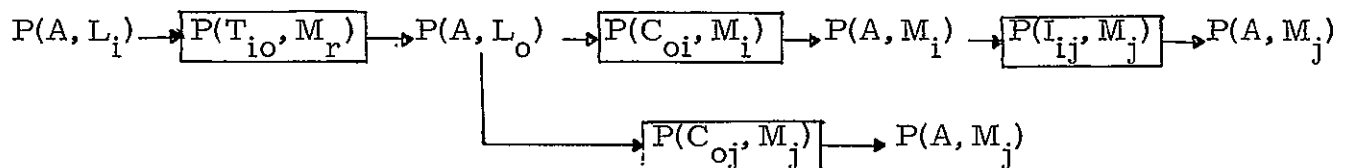
1.5 Comparison of Conversion Methods

It is obvious from the analysis so far, that the major problems of program migration lie not so much in the differences between FORTRAN dialects as in the differences between the source and host computers. Even if they used a common language, programmers writing for different computers would produce different programs.

As can be seen, each of the methods so far described has advantages and disadvantages. Method 1 is attractive from the standpoint that a common language for IPAD would impose a standard FORTRAN on all source programs. Pre-existing code would be converted to it, and all new programs would be written in it. The major disadvantage is that it is still possible to write machine dependent code in a common language, and the temptation to do so would be as strong as ever - namely, to improve performance on one's own computer. A priori, Method 2 appears to have little to recommend it. The writing of a new compiler, or the rewriting of an already existing one to provide for different dialects is on the face of it more difficult than making the conversion at the source language level. Method 3 does just that, and it should be more cost effective than Method 1. However, without a common language and an official version of each program, uniform standards of program documentation and maintenance would be difficult to enforce. Method 4 has the advantage that conversion is exact, but execution is inefficient - again there is no common language.

An interesting, but expensive possibility, denoted Method 5 provides for a common language as in Method 1, and allows as an option, the exact conversion of Method 4.

Method 5.



Here, if $d=0$, m translators, $m + n$ compilers, and mn interpreters are needed for the full system.

The advantage of a common language is expected to be decisive for IPAD users, and in what follows, the emphasis will be placed on conversion by way of Method 1, but much that will be said will apply to the other methods as well.

1.6 Implementation of Language Converters

While not strictly necessary, it is of some advantage that the conversion processes themselves be machine independent. This is not as easy to achieve as it might at first appear. Suppose it is required that all conversion programs be written in a common implementation language. Then a compiler is needed for each target computer. By hypothesis, this condition is fulfilled if FORTRAN is chosen for this purpose. But this approach must be handled carefully to avoid machine dependencies creeping into the conversion code, since it is just as easy to write machine dependent conversion programs as OMs. For example, for Method 1, the implementation process is represented by

$$P(T_{io}, L_j) \leftarrow \boxed{P(C_{jk}, M_k)} \rightarrow P(T_{io}, M_k)$$

where L_i is the source FORTRAN dialect, L_o is the common IPAD FORTRAN, and L_j is a machine independent subset of L_i (e.g., ANSI FORTRAN).

Another approach that reduces the labor of creating a special implementation language compiler makes use of a two stage process. Here a simple bootstrap compiler, C_{1k} , written either in machine independent FORTRAN, or in the assembly language of the target machine k , is compiled and then used to compile the second stage compiler, C_{2k} , written in language L_1 . C_{2k} is a macro processor for the macro language L_2 . The converter (translator, compiler, or interpreter) is written in L_2 and compiled by C_{2k} . This is essentially the method of STAGE 2 [1]. Schematically,

$$\begin{array}{l} P(C_{2k}, L_1) \rightarrow \boxed{P(C_{1k}, M_k)} \rightarrow P(C_{2k}, M_k) \\ P(T_{ij}, L_2) \rightarrow \boxed{P(C_{2k}, M_k)} \rightarrow P(T_{ij}, M_k) \end{array}$$

A macro processor provides a reasonably efficient way to resolve syntactic differences between a pair of languages, since in principle, all that is required is the substitution of one string of characters for another. The fixed part of the first string is a template representing the distributed macro name. The macro body generates a new fixed part, inserts parameters to replace those from the original string, and produces the replacement string.

Either method solves the context-free conversion problem. When a syntactic unit (sentence, statement) of the input text can have more than one meaning, depending on the context in which it is embedded, the problem becomes more difficult. By way of illustration, suppose the following subroutine was written to be executed on the CDC 6600, which is capable of storing 10 6-bit alphanumeric characters in one 60-bit word. It is desired to translate this routine for compilation and execution on a computer in which 8-bit alphanumeric characters are stored four per word.

```

SUBROUTINE PACK
C  THIS ROUTINE READ IN AN 80 COL CARD INTO AN ARRAY IN
C  WHICH THE CHARACTERS ARE STORED, ONE PER WORD, RIGHT
C  ADJUSTED WITH ZERO FILL.  THE CHARACTERS ARE THEN PACKED
C  INTO 8 WORDS, 10 TO A WORD.
  DIMENSION IN (80), IOUT (10)
  READ 1000, IN
1000  FORMAT (80 R1)
  I = 1
  DO 10 J = 1, 8
    IOUT (J) = 0
  DO 10 K = 1, 10
    IOUT (J) = IN (I) + IOUT (J) * 64
  10  I = I + 1
  PRINT 1001, IOUT
1001  FORMAT (A10)
  RETURN
END

```

The only dialect statement in the subroutine is the first format statement, yet nearly half the statements are machine dependent and, therefore, ambiguous.

The translator can detect and replace the dialect statement although the details of implementation may not be simple. For example, replace -

```
1000  FORMAT (80R1)
```

with

```
1000  FORMAT (80A1)  
      CALL PATCH (IN)
```

and insert

```
      SUBROUTINE PATCH (IN)  
      DIMENSION IPT (64), JPT (64)  
      DO 10 I = 1, 80  
      DO 5  J = 1, 64  
      IF (IN(I).EQ.IPT(J) GO TO 10  
5      CONTINUE  
10     IN (I) = JPT (J)
```

IPT and JPT are preset with a data statement to the character set left adjusted with space fill, and right adjusted with zero fill, respectively.

What the translator cannot do without considerable analysis, is detect that the indices on both DO-loops in the original program are machine dependent. The detection and conversion of machine dependent code is the central problem for automatic language translation and will be discussed more fully in subsequent sections of this report.

1.7 Preliminary Remarks on a Common Language for IPAD

Method 1 requires that all programs be translated to a common machine independent FORTRAN dialect, called here, IPADF.

At the syntax level, a language is just a set of rules for stringing the symbols of an alphabet together. At the semantic level, a language is a method for recording and communicating information. The two aspects of language are joined by the process of interpretation, whereby symbol strings generated by a set of syntax rules are assigned meanings. It is easy to see that a symbol string can be given more than one interpretation, so that fixing the syntax is not sufficient to fix the interpretation. As any student knows, there is no way to infer the meaning of a word in a strange language by just looking at it. If it can't be found in a dictionary,

its meaning must be determined by observation of the response it evokes in a user of the language.

The interpretations given FORTRAN statements vary among users. The relevant response to an interpretation is the action taken by the user's computer, and that response is determined by the compiler. Thus, it is not always possible to know what a statement in a FORTRAN program means without knowing the computer on which the program is to run and what the compiler does in translation. This is clearly unsatisfactory for IPADF whose statements are intended to be interpreted in just one way.

While it is not possible to structure a language in a way that compels the user to adopt the intended interpretation, it is possible to develop the syntax in a way that will make it easier for him to do so. It is most important that the language be rich enough so that useful concepts can be given explicit statement. This means that if it is customary for a user of one machine to interpret a FORTRAN statement differently than a second user does, the statement must be replaced with a pair of statements, one for each interpretation. Thus, it turns out that a machine independent language is of necessity richer in detail than one specialized to a single computer.

Following an examination of two well known FORTRAN dialects in Section 2, the main features of a machine independent IPAD FORTRAN are described in Section 3.

2.0 FORTTRAN SOURCE CODE TRANSLATION ON THIRD GENERATION COMPUTERS

It is expected that IPAD will be implemented first on conventional third generation computers with later transfer to fourth generation vector processors. For both Method 1 and Method 3 of the preceding section, translation from one FORTRAN dialect to another is required, either to a common language (Method 1), or between pairs of source-host dialects (Method 3). In either case, the feasibility of translation must be demonstrated.

In this section, two well known FORTRAN dialects are examined, FORTRAN Extended for the CDC 6000 series computers, and FORTRAN IV (H Extended) for the IBM 360 series computers. For convenience, the former will be called CDCF, the latter IBMF.

Neither language includes the other as a subset. Each contains statement forms peculiar to it, some denoting extensions to the basic FORTRAN language, others included to utilize certain machine features, and others representing little more than variations in syntax.

The catalogue of differences between CDCF and IBMF given below were obtained by comparing the respective reference manuals [2], [3], and do not include deviations introduced by the compilers.

2.1 Features In CDCF Not In IBMF

- (2.1.1) ENCODE-DECODE (allows data moves in main memory under format control)
- (2.1.2) Implied DO-loops in data statements
- (2.1.3) Data initialization in labelled COMMON by DATA statements outside a BLOCK DATA subprogram
- (2.1.4) Two branch arithmetic and logical IF statements
- (2.1.5) Literals and octal constants in arithmetic statements
- (2.1.6) Masking expressions (permits use of logical operators on non-logical variables)
- (2.1.7) Abbreviated subscripts on arrays, and abbreviation of logical symbols
- (2.1.8) Left and right justified literal constants with zero fill

2.2 Features in IBMF Not In CDCF

- (2.2.1) IMPLICIT type declaration by initial letter
- (2.2.2) Maximum of seven subscripts in array declarations
- (2.2.3) Data initialization in type declaration
- (2.2.4) GENERIC statement (allows function calls by generic name)
- (2.2.5) Dummy variables on ENTRY statements
- (2.2.6) List directed I/O (allows data transfers formatted by separators)
- (2.2.7) Call-by-name for subroutine variables

2.3 Syntax Differences Between CDCF And IBMF

	<u>Element</u>	<u>CDCF</u>	<u>IBMF</u>
(2.3.1)	comment	C, *, or \$ in col. 1	C in col. 1
(2.3.2)	string delimiter in FORMAT statements	*	'
(2.3.3)	maximum name length	7 characters	6 characters
(2.3.4)	alphabetic symbol set	A - Z	A - Z, \$
(2.3.5)	PAUSE n	n (octal)	n (decimal)
(2.3.6)	END	termination program	does not terminate program
(2.3.7)	TYPE in type declarations	optional	no
(2.3.8)	computed GO TO out of range	abort	CONTINUE
(2.3.9)	assigned GO TO out of range	go to absolute address	abort
(2.3.10)	RETURNS declaration	SUBROUTINE SUB, RETURNS (A, B)	SUBROUTINE SUB (*,*)
(2.3.11)	Complex argument in IF statement	real part taken	not allowed
(2.3.12)	PROGRAM statement	yes	no
(2.3.13)	NAMELIST	\$ NAME \$	& NAME & END

2.4 Machine Dependent Statements In CDCF And IBMF

- (2.4.1) type ECS in CDCF (identifies variable in Extended Core Storage)
- (2.4.2) type *S statements in IBMF
- (2.4.3) Any other FORTRAN statement in which the value of a parameter depends on the word length of the machine, the collating sequence, or the way in which memory is organized.

2.5 Input/Output And Data Transfer Statements In CDCF And IBMF

(2.5.1) Asynchronous read-write

CDCF

BUFFER IN (a,p) (A,B)

BUFFER OUT (a,p) (A,B)

IF (UNIT(a))r,s,t

a is data set number, p is parity mode, A,B first and last address in main memory; r,s,t are respectively, unit ready, end-of-file detected, parity error.

IBMF

READ (a,ID=n) list

WRITE (a,ID=n) list

WAIT (a,p) list

a is data set number, n is identifier, list defines area in main memory, and is either an array name or first and/or last address. If list is not specified on READ, a record is skipped.

(2.5.2) Random access

CDCF

OPENMS (a,ix,d,p) open mass storage file

READMS (a,fwa,n,i) read mass storage file

WRITMS (a,fwa,n,i) write mass storage file

STINDEX (a,ix,l) store index

a is data set number, ix is first word address of index, l is index length, p is parity mode, fwa is first word address of record, n is number of words to be transferred, and i is the record number or address of record number (name).

IBMF

DEFINE FILE a (m,r,f,v),...

READ (a'r,b,ERR=c) list

WRITE (a'r,b,ERR=c) list

FIND (a'r)

where a is data set number, m is number of records in data set, r is maximum size of each record in a, f is the format control, and v is the pointer to the record involved in the current operation.

2.6 Implementation Restrictions In IBMF And CDCF

There are no uniform standards restricting the range of most FORTRAN parameters. As a result, part of the definition of the language is left to implementive decisions which may or may not be documented. For example, in IBMF, the following restrictions are imposed:

- (2.6.1) DO-loop nesting is limited to 25 levels
- (2.6.2) The test value in a DO-statement may not exceed $2^{31} - 2$
- (2.6.3) No more than 255 characters allowed in a literal constant or in a field in a FORMAT statement
- (2.6.4) Up to 50 nested references to another statement function can be made in the definition of a statement function.

In CDCF,

- (2.6.5) DO-loop nesting can extend to 50 levels
- (2.6.6) The test value in a DO-statement may not exceed 2^{15} .
- (2.6.7) A literal in an expression is limited to 10 characters; in a DATA statement, the length cannot exceed 19 continuation cards; in a FORMAT statement, 136.
- (2.6.8) The number of nested references to other statement functions is undocumented; up to 63 arguments are allowed.

2.7 CDCF And IBMF Interfaces With Job Control Language

There exists a close relationship between a version of FORTRAN in use with a given operating system, and that system's job control language. In a general way, the algorithm is described in FORTRAN, and the manner of execution in a given computer system is specified in the job control language. The distinction is not precise, and in some systems run time specifications are made in FORTRAN that in other systems are given in the job control language. In CDCF, for example, overlays can be defined and called explicitly in the program. In the IBM system, overlays are defined in the job control language, and called implicitly in the program. Again, CDCF requires that the names of all input/output files be given in a special PROGRAM statement. The IBM system provides this information in the job control language.

2.8 Translation From IBMF To CDCF

Now suppose it is desired to translate programs written in IBMF to CDCF. For this case, items under 2.1 above play no role, except perhaps for optimization. Items under 2.3 are the most common cause of trouble whenever an attempt is made to run an IBMF program on a CDC computer without preliminary editing, but they are in fact easy to translate since each can be converted as encountered.

Those features of IBMF having no precise counterparts in CDCF listed under 2.2 are more difficult to translate. For example, the IMPLICIT type declaration, (2.2.1) IMPLICIT INTEGER A-H requires in CDCF that each variable beginning with one of the letters A through H be included in a type INTEGER declaration.

Arrays having more than three subscripts (2.2.2) must be converted for CDCF. One way to deal with this problem is to define a function subprogram with the same name as the array, that returns the specified array element from the original array, renamed as a one dimensional array. For example, in IBMF

```
DIMENSION A(5,10,20,3)
      :
      X = A(I, J, K, L)
```

would be translated to

```
COMMON/B/B(3000)
      :
      X = A(I, J, K, L)
```

where A is defined by

```
FUNCTION A(I, J, K, L)
COMMON/B/B(3000)
M = I - 1 + 5 * (J - 1 + 10 * (K - 1 + 20 * (L - 1)))
A = B(M)
RETURN
END
```

For CDCF data initialization must be removed from type declarations (2.2.3) and put in a separate DATA statement.

GENERIC statements (2.2.4) must be deleted for CDCF, and the appropriate subprogram name inserted in all function and subroutine calls.

IBMF allows a parameter list on ENTRY statements (2.2.5); CDCF does not. This problem can be solved in several ways. One solution that leaves the subroutine calls undisturbed is illustrated below.

IBMF code

```
SUBROUTINE SUB (W, X, Y, Z)
  (body 1)
  ENTRY SUBA (Y, X, U)
  (body 2)
END
:
CALL SUB (A, B, C, D)
:
CALL SUBA (B, A, E)
```

CDCF code

```
SUBROUTINE SUBT (W, X, Y, Z, U)
  (body 1)
  ENTRY SUBTA
  (body 2)
RETURN
END
SUBROUTINE SUB (W, X, Y, Z)
  CALL SUBT (W, X, Y, Z, U)
RETURN
END
SUBROUTINE SUBA (Y, X, U)
  CALL SUBTA (W, X, Y, Z, U)
RETURN
END
```

List directed I/O (2.2.6) will have to be handled by a list scanning subroutine.

Call-by-name for subroutine variables (2.2.7) is taken care of in CDCF either by the LOC library function, or by making the variable a one dimensional array.

Real numbers in the IBM 360 are expressed in base 16 (hexadecimal notation). All have a 2-digit exponent. The characteristics for REAL*4 (single precision), REAL*8 (double precision) and REAL*16 (extended precision) are 6, 14, and 28 digits respectively. For most mathematical work, satisfactory precision is preserved if REAL*4 and REAL*8 variables are represented as single precision (REAL) variables in CDCF, while REAL*16 are classified as double precision. No difficulty should arise if LOGICAL*1 and LOGICAL*4 are replaced by LOGICAL; similarly for INTEGER*2, and INTEGER*4. Hexadecimal constants, on the other hand, can be expected to be involved in machine dependent operations, and will require manual translation.

The input/output and data transfer statements of 2.5 are more subtly machine and software system dependent. For example, the asynchronous read/write statements (2.5.1) in the two languages are quite similar and, except for the record skipping feature in IBMF, translation involves little more than syntax changes. It is not clear, however, that a CDC programmer and an IBM programmer would resort to asynchronous operation at the same place and under the same circumstances in their respective programs. For the present, it will be assumed that the translator makes the translation, but inserts a warning flag that all might not be well.

The situation with regard to the random access statements is similar, except that here translation is complicated further by the two languages specifying different parameters as well as different formats. Those statements probably should be flagged for manual translation.

The conclusion that emerges from the above thicket of detail is that, while not always easy, machine translation of FORTRAN programs from one dialect to another is possible, provided that the translator can detect that translation is required.

Machine dependent data and parameter values are most likely to occur in

- DATA statements
- FORMAT statements
- IF statements
- DO statements

integer assignment statements
logical assignment statements

and can be expected to require the most attention from the programmer to resolve.

The translation process as it is presently conceived, proceeds in the following way. All programs are submitted with test data and the expected output. An attempt is first made to compile and execute the program without preliminary translation to determine adherence to IPAD programming standards. If the program fails to compile correctly, a preliminary examination of the error listing should be made to determine if translation will resolve the difficulty. (The error may be an untranslatable statement.)

If, after translation, the program compiles but fails to execute correctly, it can be assumed that the program contains machine dependent code. At this point, the programmer takes over, and begins a detailed examination of the suspected parts of the code aided by the IPAD compiler's cross reference listing. His editing comments are keypunched to create a correction deck for the program in UPDATE (editor) format. The edited code is compiled, execution is attempted, and the cycle repeats until a correct executable program is obtained.

3.0 INTRODUCTION TO THE DEVELOPMENT OF A MACHINE INDEPENDENT FORTRAN

3.1 Preliminary Remarks

One conclusion that emerges from an examination of both CDCF and IBMF is that neither language is appropriate for use as the machine independent IPAD FORTRAN (IPADF). Both languages contain explicit machine dependent statement forms and, more important, statement forms they share in common are often given different interpretations by the respective compilers.

The proliferation of FORTRAN dialects, of which the above two are examples, is caused by a number of factors, not the least of which is the "not-invented-here" syndrome. A less frivolous reason is that computer language development has been directed toward incompatible goals. While the language theorists sought to develop languages suitable for the machine independent description of algorithms, another and more influential group, wanted languages for practical programming use that allow the programmer to reduce the amount of detail he need write, and yet produce programs that execute satisfactorily on predetermined computers. The result was an odd compromise. The theorists specified nothing quantitative in languages designed primarily for numerical calculations. In fact, in the hope of preserving machine independence, they defined nothing that was not, in some sense, common to all machines. The developers of practical languages merely added what they felt they needed either, explicitly to the syntax, or implicitly in the interpretation given by the compiler.

The error of the theorists was in leaving so much undecided. What is not specified is left to the user to interpret as he sees fit, and each will avail himself of the freedom granted. A truly machine independent language will require more, not less, information than one intended for a single class of computer. For example, if the FORTRAN statement, $C = A + B$, is interpreted routinely as a 24-bit operation on one machine, and as a 48-bit operation on another, the two computers are very possibly not working the same problem. The numerical precision with which an arithmetic process is carried out is generally crucial to its accuracy, and it is a fundamental shortcoming of a machine independent language if it is incapable of expressing this most basic of parameters.

In IPADF it is expected that the precision of real numbers is to be specified by an explicit statement of the number of significant decimal digits desired; thus, REAL*s, has s significant digits. There is to be no default option, and hence no REAL or DOUBLE PRECISION declaration. No penalty is expected if calculations are carried out with higher precision than required. The treatment of complex numbers is similar, and needs no separate discussion.

The declaration INTEGER*s specifies the number of decimal digit positions required, and is included for the sole purpose of aiding the compiler in determining storage requirements.

Integers expressed to bases other than 10 constitute a special class, denoted by the declaration BASE*n.s, where n is the base and s is the number of base n digits. Logical constants have a fixed length and require no additional specification.

A similar sort of vagueness plagues most FORTRAN dialects with respect to their character set, or alphabet. Few dialects have common alphabets, or order them the same way, ANSI standards notwithstanding. Even when the sets more or less agree, they are often partitioned into subsets differently. For example, IBMF counts "\$" as an alphabetic character; CDCF, as a special symbol. Clearly, IPADF must either (i) specify once and for all what its character set is, how the symbols are to be classified, and in what order they are to be ranked (collating sequence), or (ii) this information must be provided explicitly with each program requiring it. It is expected that the first of these alternatives will be adopted for IPADF.

Packing and masking operations occur frequently in FORTRAN programs, and almost invariably are machine dependent involving an addressable unit, usually a machine word, capable of storing n characters of the alphabet, where n is a machine dependent parameter. It seems likely that the requirements for machine independence will make it necessary to introduce a type LITERAL*s declaration to define the number of characters in the addressable unit.

Because every variable must be given an explicit length declaration in IPADF, the default rules for implicit typing must be dropped. However, the IBMF

IMPLICIT type *s declaration will be retained to provide blanket typing, except that the length specification s, optional in IBMF will be mandatory in IPADF.

It is tempting to consider a general overhaul of FORTRAN mixed mode arithmetic for IPADF, since different dialects often produce different results for simple arithmetic statements involving mixed real and integer operands. At the very least, arithmetic operations involving non-numeric variables should be disallowed.

For the IPAD environment, it is undesirable that run-time specifications appear in the body of a program. Specification of overlays, for example, would have to be deleted, or ignored, if the program is to execute on a virtual address computer. Leaving aside development of a machine independent job control language as visionary at present, it would appear that the best solution for IPAD is to require that all run-time information be provided on printed forms for operator use in constructing the job control deck for the host computer.

The above discussion applies primarily to so-called third generation computers in which instructions usually involve one operation per instruction, the most notable exception being block transfers of data. Fourth generation computers are distinguished by a capability to perform arithmetic and logical operations on all or a selected subset of the elements in a one dimensional array or vector.

It turns out that very little modification of the FORTRAN language is required to permit elementary vector operations. The usual convention is that the name of a one dimensional array represents the array in an arithmetic or logical expression, while a vector in a higher dimensional array is selected by replacing the index of its elements by a * in its indexed name. For example, if DIMENSION A(10,20), B(10) then $B = A(*, 5)$ means that the fifth column of A is transferred to the vector B.

A proposed machine independent IPAD language incorporating the features discussed above is sketched briefly in the following paragraphs.

3.2 IPAD FORTRAN (IPADF)

The character set of IPADF are the 26 letter symbols A-Z, the 10 digit symbols 0-9, and the special symbols = + - * / () , . \$; Letter and digit symbols are called collectively alphanumeric symbols.

Names begin with a letter and can contain up to 7 alphanumeric symbols. Variables are identified by name and declared by type and length. The following types are recognized by IPADF: REAL, COMPLEX, INTEGER, LOGICAL, BASE, LITERAL. For real and complex numbers, the length is the minimum number of significant decimal digits in the fraction part of its floating point representation. Logical variables are of constant length and require no specification. For all the remainder, the length specifies the maximum number of elements denoted by the variable, for integers, decimal digits; for base variables, the number of base a digits; and for literals, the number of characters. Type REAL, COMPLEX, INTEGER, and LITERAL follow the form

type *s name₁, name₂, ...

where s is the length. Based variables are declared by

BASE *n.s name₁, name₂, ...

where s is the maximum number of base n variables (octal, hexadecimal, etc.)

For logical variables

LOGICAL name₁, name₂, ...

is sufficient.

Arrays may have up to 7 dimensions and are declared in a dimension statement of the form

DIMENSION Array name (d₁, d₂, ..., d_n)

Arrays can be referenced in a number of ways. An array name, standing alone, denotes the whole array. An array name followed by coordinate variables, $A(I, J, K)$ picks out an element in the array, while $A(*, J)$ designates the J th column vector in the array A .

There are no default type declarations. However, the IMPLICIT declaration allows variables to be typed by their initial letter. The statement form is

IMPLICIT type *s

followed either by the specific letters involved, or by their range (e.g. I-N).

Arithmetic operations involving variables of different type is called "mixed mode arithmetic." The results of valid IPADF mixed mode arithmetic operations are shown in Table (3.1) below.

	Real	Complex	Integer	Logical	Base n	Literal
Real	Real	Complex	Real	----	Real	----
Complex	Complex	Complex	Complex	----	----	----
Integer	Real	Complex	Integer	----	----	----
Logical	----	----	----	Logical	----	----
Base n	Real	----	----	----	Base n	----
Literal	----	----	----	----	----	----

Table (3.1) Mixed Mode Arithmetic

Definition of IPADF assignment statements follow normal FORTRAN rules except for the restrictions imposed on mixed mode arithmetic shown above, and the extension of the notion of variable to include vectors as well as scalars. Thus, if A , B , and C have been dimensioned as N element one dimensional arrays, $C = A+B$, has the same meaning as:

```

DO 1 I = 1, N
1  C(I) = A(I) + B(I)

```

Run time declarations are not permitted in IPADF. Directions for segmenting memory, definition of overlays and input/output files, etc. are relegated to the job control language.

In general features common to both CDCF and IBMF are retained in IPADF. For unshared features, the decision on which to incorporate and which to reject is based on estimated ease of coding in a machine independent environment, a purely subjective judgement. This list below is not complete, but covers most of the differences between CDCF and IBMF noted in Section 2.

- (3.2.1) ENCODE-DECODE is not implemented
- (3.2.2) Implied DO-loops in DATA statements are permitted
- (3.2.3) Data initialization in labeled COMMON is allowed outside a BLOCK DATA subprogram
- (3.2.4) Two branch arithmetic and logical if statements are not allowed
- (3.2.5) Literal and Base n constants can appear in arithmetic statements subject to the rules of mixed mode arithmetic
- (3.2.6) Masking expressions are not permitted
- (3.2.7) Abbreviated subscripts on arrays, and abbreviated logical symbols are not permitted
- (3.2.8) Left and right justified literal constants are permitted, but with blank fill.
- (3.2.9) Data initialization is confined to DATA statements
- (3.2.10) No GENERIC statement
- (3.2.11) Dummy variables are permitted on ENTRY statements, following IBMF rules
- (3.2.12) List directed I/O not implemented

3.3 FORTTRAN Dialect to IPADF Translation

For the implementation of Method 1, it is necessary that OM's written in each of the m FORTRAN source dialects, L_i , be translated into L_o , the common IPAD FORTRAN (IPADF).

It will be assumed that the translator T_{io} is written in L_i , though, of course, this is not necessary; any language at the source installation will do. The main point is that each source user is expected to develop and checkout his own translator, and convert his own OM's. The complete process is depicted schematically below.

- (1) compile translator on source computer

$$P(T_{io}, L_i) \longrightarrow \boxed{P(C_{ii}, M_i)} \longrightarrow P(T_{io}, M_i)$$

- (2) translate test program A

$$P(A, L_i) \longrightarrow \boxed{P(T_{io}, M_i)} \longrightarrow P(A, L_o)$$

- (3) compile test program on source computer

$$P(A, L_i) \longrightarrow \boxed{P(C_{ii}, M_i)} \longrightarrow P(A, M_i)$$

- (4) execute test program on source computer

$$D \longrightarrow \boxed{P(A, M_i)} \longrightarrow R_S$$

- (5) compile translated test program on host computer

$$P(A, L_o) \longrightarrow \boxed{P(C_{oj}, M_j)} \longrightarrow P(A, M_j)$$

- (6) execute test program on host computer

$$D \longrightarrow \boxed{P(A, M_j)} \longrightarrow R_H$$

- (7) compare results R_S and R_H from steps (4) and (6) respectively.

The validation process was described in some detail to direct attention to the fact that final certification of each translator will have to be made on a host computer. Initial translator checkout and translation on source computers is

proposed here to distribute the workload more evenly, but this approach involves considerable travel between source and host installations until checkout is completed. The alternative is to perform all translation on the host computers. If the workload permits, this may be the better way. A final decision cannot be made at this time.

Translation from source dialects to IPADF is mainly a matter of removing machine dependent code from OM's. Some of this code can be expected to satisfy the syntax rules for both languages, and will be impossible for a translator to detect. An example of this type was given in Section 1.6.

Each installation is responsible for its own OM translation, and, consequently, is free to develop whatever aids it chooses for the purpose. However, it is expected that most will find it useful to incorporate into the translator the capability for flagging sections of code having a high probability of being machine dependent. Particular attention should be given to:

- (3.3.1) DO-loops involving integer variables
- (3.3.2) Logical IF statements
- (3.3.3) Non-standard library functions operating on characters
(e.g., SHIFT, PUT, GET, etc.)
- (3.3.4) DATA statements containing binary, octal or hexadecimal data
- (3.3.5) FORMAT statements (non-standard, literal)

Hand corrections to programs will normally be made via the installation's standard editing program (e.g., CDC UPDATE). Final validation of an OM will require comparison of its output with that obtained from the untranslated program, and for some cases, it can be expected that several passes through the translate-correct-edit loop will be necessary.

3.4 IPAD Implementation Language

Implementation strategies were discussed briefly in Section 1.6, and the conclusion was reached that the implementation language for IPADF should itself be machine independent. One language description will suffice then for all IPAD users involved in the development of IPADF, and if the implementation language is one for which a compiler is generally available, it remains only to determine

its suitability for the task.

The main operations performed by a compiler are (i) parsing of the statement symbol strings to separate the fixed and variable parts and establish the precedence of operations, (ii) conversion of constants and data to internal machine representation, (iii) allocation of storage to variables by name and type, and finally (iv) generation of the machine language code.

There is no obvious reason why algorithms performing each of those operations cannot be described satisfactorily in FORTRAN, augmented, as necessary, by the addition of library subprograms. Call this implementation language, IMPF. The principal advantage of this approach is that, if care is taken to keep machine dependent code out of the program itself, IPADF and its successor for fourth generation computers, IPADFV, can be written in IMPF for each host computer and compiled by its standard FORTRAN compiler.

It is not intended that the compiler itself be machine independent. Much of the code could be made so, but at exorbitantly high cost. Very likely it would be necessary to settle on FORTRAN character as the information unit, and store data one character per machine word. Every variable name in the symbol table would then be represented by an array, named by a pointer variable. Storage allocation strategies would have to be made uniform, and this could cause undesirable system repercussions. Then, too, it would be desirable to translate the FORTRAN statements to an intermediate language so that the only machine dependent operation was the final generation of machine code. It is unlikely that very efficient compilers would result from an approach subject to so many restraints.

4.0 MIGRATION OF OM'S FROM THIRD GENERATION TO FOURTH GENERATION COMPUTERS

4.1 Introduction

A rudimentary capability for expressing vector operations is incorporated in the proposed design for IPADF. More extensive features are required for fourth generation computers such as Control Data's STAR-100, which can perform arithmetic and logical operations on character and bit strings, as well as normal and compressed (sparse) vectors.

For concreteness, STAR will be taken as representative of fourth generation computers, and a brief sketch of its salient features will be useful.

STAR is a virtual address computer able to distinguish 2^{48} distinct address bits. These addresses are automatically converted by the instructions using them into bit, byte, half-word or full-word addresses. The central memory consists of 500,000 full words of core storage, and a high speed register file of 256 words. Arithmetic operations are carried out on either 32-bit half-words or 64-bit full-words in either scalar or vector mode.

Vector operations can be performed on normal or sparse vectors. A normal vector is just an ordered list of half-words or full-word elements. A sparse vector is a vector formed by application of a binary mapping function, or order vector, that extracts in order a subset of elements from the original vector. When arithmetic operations are performed on sparse vectors, these elements of the original vector not appearing in the sparse vector are taken to be zero.

In addition to the standard arithmetic and logical operations, a comprehensive set of macro and APL functions (c. f. Iverson⁵) are implemented such as contract, expand, merge, mask, element sum, element product, maximum element, minimum element, vector dot product, search, and select.

4.2 IPADF Extended For Vector And String Processing (IPADFEV)

Part of the usefulness of FORTRAN is that programs in it are shorter and more readable than if written in a computer's assembly language. To preserve this feature for fourth generation computers, it will be necessary to extend the

capabilities of the language to encompass the more sophisticated instruction repertoire of these machines. This can be done in two ways: by enlarging the syntax, or by creating new in-line functions.

In fact, both methods will be found to be useful. Certainly, new declarations will be needed to accommodate the new variable types, and logical functions extended over all the members of an ordered set will require quantifiers in addition to the standard Boolean functions. On the other hand, many of the macro and APL instructions in STAR can be implemented easily and efficiently by in-line functions. Probably, frequency of use and readability should be the criteria for deciding whether an operation should be represented by an in-line function or by a new statement form. Elements in a function argument list tend to be anonymous, particularly when they can be distinguished only by their position in the list. The alternative is not always better - witness the bewildering array of infix operators in APL.

Fourth generation computers are distinguished from third generation machines primarily by their capability for processing lists efficiently. In the computer, a list is just a consecutive set of storage locations, and is defined by a starting location and the number of elements contained in it. From the programmer's point of view, the meaning assigned a list depends on how it is to be used. For example, a list A can represent (i) an n-dimensional vector A (with ith component A_i) which enters as single variable in a calculation $f(A, b, c, \dots)$ or (ii) the set of values of a single variable a, whose typical element a_i is the value of a in the ith calculation of $f(a_i, b, c, \dots)$ for $i = 1, 2, \dots, n$. Often a list contains logically distinct sublists which are extracted to become new lists in subsequent operations.

The case for which the elements of a sublist form an arithmetic progression in the master list occurs often enough to justify a special notation of the form

$$B = A(M_1 : M_2 : M_3)$$

where B is composed of those elements of A whose indices i satisfy the inequality $M_1 + M_3(i - 1) \leq M_2$. More generally, IPADFV allows any of the following forms to apply as well to the component vectors of an array.

- (i) $M_1 : M_2 : M_3$
- (ii) $M_1 : M_2$
- (iii) $*$
- (iv) $M_1 : * : M_3$
- (v) $M_1 : *$

Here M_1 and M_3 have the value 1 when omitted. The symbol $*$ denotes that M_2 has the value of the dimension length. For example, if

DIMENSION A(20), B(10,6)

then

A(3:18:7) represents A(3), A(10), A(17)

B(2:*:6, 4) represents B(2, 4), B(8, 4)

Reference to sparse vectors in IPADFV have the general form $[V, L]$, where V is the vector from which the elements were obtained, and L is a logical vector specifying which elements of V occur in $[V, L]$. Subscripts can appear on either V or L, and follow the rules given above. For example,

$[V(I:J), L(50:100)]$

The logical operators

.NOT.	negation
.OR.	alteration
.XOR.	disjunction
.AND.	conjunction

of IPADF which can take either vector or scalar arguments is extended in IPADFV by the unary logical vector quantifiers

.ALL.	L	universal conjunction of all elements of L
.ANY.	L	existential alternation of all elements of L
.NONE.	L	universal negation denial of all elements of L

Logical expressions are evaluated by scanning from left to right with precedence of logical connectives established by

.NOT.
 .ALL. .ANY. .NONE.
 .AND.
 .OR. .XOR.

If L is a scalar logical variable, the .ALL. L and .ANY. L reduce to L, and .NONE. L reduces to .NOT. L.

A relational assignment statement is introduced in IPADFV of the form

P = Q. R. S

where Q and S are vectors, and R is one of the relations

.EQ. .NE. .GE. .LT.

Execution of this statement leads to three different results depending on the type of P. If P is an integer variable, P is assigned the index of the first elements of Q and S to satisfy the relation R. If P is an integer vector, each element of Q is compared with successive elements of S, and whenever the relation R is first satisfied, the element of P corresponding to the element of Q is assigned the index of S. If P is a logical vector, each element of P is set to .TRUE. if the corresponding elements of Q and S satisfy R; otherwise, .FALSE.

The scalar (dot) product C of two vectors A and B is simply

C = A*B

The notation

B = +A
 and C = *A

where A is a vector and B is a scalar, denote the sum and product of the elements of A, respectively.

In addition to the extensions of the IPADF syntax noted above, the set of in-line functions is enlarged in IPADFV to include most of the STAR macro and APL-like instructions. These include:

MERGE (A, Z, B) selects an element from vector A or vector B, depending on whether the corresponding element value in logical vector Z is .TRUE. or .FALSE. No elements of A or B are skipped.

MASK (A, Z, B) selects an element from vector A or vector B, depending on whether the corresponding element value in logical vector Z is .TRUE. or .FALSE. The element not selected is passed over. The symbol * in either vector position indicates that no selection is made if the element normally selected would have come from the vector in that position. For example

MASK (A, Z, *) is equivalent to selecting an element of A whenever the corresponding value of Z is .TRUE., while (*, Z, B) selects an element of B whenever the corresponding value of Z is .FALSE.

4.3 IPADF To IPADFV Translation

The principal problem for IPADFV, the fourth generation FORTRAN for IPAD is the same as for its predecessor, IPADF, namely to preserve machine independence. Clearly, the introduction of list parameters (vectors, strings, etc.) in fourth generation computers does not of itself aggravate the situation unduly since the same length prescriptions can be applied to the elements of a list as to individual variables. What must be taken into account is the near certainty that during the lengthy transition phase from third to fourth generation processors, the IPAD host computers will be a mixed bag - some belonging to one class and some belonging to the other. Nevertheless, if machine independence is to be preserved, the OM's written in this period should execute on all host machines, though it is unreasonable to expect them to execute on all with the same efficiency. This concern for linking third to fourth generation computers was the prime reason for introducing elementary vector operations in IPADF. For a third generation machine, the definitions are little more than abbreviations for simple DO loops. For a vector processor, each represents a basic machine operation.

It follows from considerations such as the above that the shift from IPADF to IPADFV should be delayed until fourth generation machines are IPAD standards.

As the migration from third to fourth generation processors progresses, it will become necessary to decide in the case of individual OM's whether to reprogram or not. The incompatibility in structure between vector and conventional computers is reflected in the programs written for them.

An example illustrating this point is the discrete Fourier transform defined by

$$(4.1) \quad f(u) = \sum_{v=0}^{n-1} g(v) \exp(-2\pi i uv/n) \quad u = 0, 1, \dots, n-1$$

where $f(u)$ and $g(v)$ are suitably chosen complex functions of the integer variables u and v , and is a basic tool in the analysis of complex wave forms, so that much effort has gone into the search for efficient procedures for making the calculations. The most successful of these have come to be called Fast Fourier Transform (FFT) algorithms, and all depend upon some form of factorization of the exponent.

When the sample size n is a power of 2, $n=2^m$, the execution time on a digital computer is reduced in this way to approximately the fraction m/n of that required by direct calculation.

The STAR algorithm consists of two parts. In part 1, the $n/2$ sine and cosine terms are calculated by an application of the polynomial evaluate (DE) instruction which performs a power series expansion

$$y_k = a_0 + a_1 x_k + a_2 x_k^2 + \dots + a_r x_k^r \quad k = 0, 1, \dots, n-1$$

Part 2 is the main loop, executed m times. The calculations of (4.1) are carried out by 10 instructions operating on vectors of length $n/2$ followed by a merge of the upper and lower halves of the real and imaginary components, respectively. This step is required to position elements $x(0)$, $x(1)$ and $y(0)$, $y(1)$ $n/2$ positions apart for the next stage. That it does so is easily seen, since at stage k , the matching elements for stage $k+1$ are $n/4$ positions apart. The merge moves all elements a_k , $k < n/2$, to a_j^1 and all elements a_k , $k \geq n/2$ to a_j^1 ,₁

where $j = 2k \pmod{n} - k \pmod{2^{s-1}}$, and $j_1 = j + 2^{s-1}$. Then if, $j = i + n/4$ it follows in either case that $j' - i' = n/2$.

Finally, a compression instruction followed by a merge instruction replaces

every other group of length s in the sine and cosine lists by the group just preceding it.

Besides eliminating the final sort required by most FFT algorithms, the procedure described above eliminates memory bank conflicts when (as in STAR) the number of banks is a power of 2.

The main loop of the FFT program written in IPADFV is given in Figure 4-1 below. All functions are in-line and represent single STAR instructions.

```
100      U0 = Y0 + Y1
          U1 = Y0 - Y1
          Y0 = X0 + X1
          Y1 = X0 - X1
          X0 = Y1 * COS
          X1 = Y1 * SIN
          E = U1 * SIN
          Y1 = X0 + E
          E = U1 * COS
          U1 = X1 + E
          X = MERGE (Y0, Z, Y1)
          Y = MERGE (U0, Z, U1)
          U = MASK (COS, Z, *)
          COS = MERGE (U, Z, U)
          U = MASK (SIN, Z, *)
          SIN = MERGE (U, Z, U)
          Z = BIT (Z, Z)
          N = N - 1
          IF(N) 110, 110, 100
110      CONTINUE
```

Figure 4-1. FFT Main Loop (Vector)

A FORTRAN version for the same algorithm, but for a conventional computer is given in Figure 4-2. The above subject was discussed at some length to emphasize the critical point that, while software migration from local FORTRAN dialects to IPADF (or its extension, IPADFFV) will permit execution on either third or fourth generation host computers, hand reprogramming will be required if the full potential of a vector computer is to be realized.

```

      DO 2 PASS=1,N4POW
      NXTLTH=2**(N2POW-2*PASS)
      LENGTH=4*NXTLTH
      SCALE=6.283185307/FLQAT(LENGTH)
      DO 2 J=1,NXTLTH
      ARG=FLOAT(J-1)*SCALE
      C1=COS(ARG)
      S1=SIN(ARG)
      C2=C1*C1-S1*S1
      S2=C1*S1+C1*S1
      C3=C1*C2-S1*S2
      S3=C2*S1+S2*C1
      DO 2 SEQLOC=LENGTH,NTHPOW,LENGTH
      J1=SEQLOC-LENGTH+J
      J2=J1+NXTLTH
      J3=J2+NXTLTH
      J4=J3+NXTLTH
      R1=X(J1)+X(J3)
      R2=X(J1)-X(J3)
      R3=X(J2)+X(J4)
      R4=X(J2)-X(J4)
      I1=Y(J1)+Y(J3)
      I2=Y(J1)-Y(J3)
      I3=Y(J2)+Y(J4)
      I4=Y(J2)-Y(J4)
      X(J1)=R1+R3
      Y(J1)=I1+I3
      IF(J.EQ.1) GO TO 1
      X(J2)=C1*(R2+I4)+S1*(I2-R4)
      Y(J2)=-S1*(R2+I4)+C1*(I2-R4)
      X(J3)=C2*(R1-R3)+S2*(I1-I3)
      Y(J3)=-S2*(R1-R3)+C2*(I1-I3)
      X(J4)=C3*(R2-I4)+S3*(I2+R4)
      Y(J4)=-S3*(R2-I4)+C3*(I2+R4)
      GO TO 2
1     X(J2)=R2+I4
      Y(J2)=I2-R4
      X(J3)=R1-R3
      Y(J3)=I1-I3
      X(J4)=R2-I4
      Y(J4)=I2+R4
2     CONTINUE

```

Figure 4-2. FFT Main Loop (Scalar)

REFERENCES:

1. The Mobile Programming System: STAGE 2, W. M. Waite, Comm. ACM vol. 13, no. 7, July 1970
2. FORTRAN Extended Reference Manual 6400/6500/6600 Computer Systems 60176600 Rev. E., Control Data Corporation
3. IBM System/360 and System/370 FORTRAN IV Language 9th Edition, GC28-6515-8 International Business Machines Corporation
4. The Use of An Algebraic Language as Both a Source and Target Language, P. H. Knowlton, Proc. 23rd Nat. Conf. ACM, 1968
5. A Programming Language, Kenneth Iverson, Wiley, 1962